Open Geospatial Consortium

Date: 2011-01-04

Reference number of this document: 09-156r2

Category: **OGC®** Engineering Report

Editor: Luis Bermudez

OGC® Ocean Science Interoperability Experiment

Phase II Report

Warning

This document is not an OGC Standard. This document is an OGC Public Engineering Report created as a deliverable in an OGC Interoperability Initiative and is not an official position of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements.

Document type:	OpenGIS® Engineering Report
Document subtype:	NA
Document stage:	Approved for release
Document language:	English

Abstract

This OGC Engineering Report documents the work performed by the participants of the Ocean Science Interoperability Experiment Phase II. This work is a follow-on to the OGC Oceans IE Phase 1 activity. Specifically, this IE addressed the following tasks:

- Automated metadata/software installation via PUCK protocol.
- Offering of complex systems (e.g. observations systems containing other systems) such as collection of stations.
- Linking data from SOS to out-of-band offerings.
- Semantic Registry and Services.
- Catalogue Service-Web Registry.
- IEEE-1451/OGC-SWE harmonization

As a result of this experiment, a number of recommendations and conclusions were identified.

Keywords

OGC, Oceans, SOS, SensorML, SWE, PUCK, IEEE-1451

Table of Contents

OpenGIS® Ocean Science Interoperability Experiment II

1. Introduction

1.1 Summary and Scope

The Oceans Science Interoperability Experiment phase II is intended to consolidate a portion of the Ocean-Observing community on its understanding of various OGC specifications, solidify demonstrations for Ocean Science application areas, harden software implementations, and produce a candidate OGC Best Practices document that can be used to inform the broader ocean-observing community. To achieve these goals, the Oceans IE will engage the OGC membership to assure that any community recommendations coming from the Oceans group will properly leverage the OGC specifications.

Potentially, Change Requests on OGC Specification will be provided to the OGC Technical Committee to influence the underlying specifications. It is not anticipated that this IE will develop any new specifications.

The OGC members that are acting as initiators of the Interoperability Experiment are:

- Southeastern Universities Research Association (SURA)
- National Oceanographic and Atmospheric Administration (NOAA)
- Texas A&M University – Academy for Advanced Telecommunications (TAMU)
- National Center for Atmospheric Research (NCAR)
- The Monterey Bay Aquarium Research Institute (MBARI)
- Gulf of Maine Ocean Observing System (GoMOOS)

The participants are also part of the OOSTethys project. Documentation about OOSTethys and the Ocean Science Interoperability Experiment, and tools such as reference implementations toolkits are available at http://www.oostethys.org.

1.2 Foreword

This is an informative document that describes lessons learned and best practices from using OGC and W3C standards. This document is not an OGC or W3C standard.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium Inc. shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

1.3 Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Name	Organization
Luis Bermudez	Southeastern Universities Research Association
David Coyle	USGS
Carlos Rueda	Monterey Bay Aquarium Research Institute
Eric Bridger	Gulf of Maine Ocean Observing System
Tom O'Reilly	Monterey Bay Aquarium Research Institute
Manil Maskey	University of Alabama
Eric Delory	dBscale Sensing Technologies

1.4 Revision history

Date	Release	Editor	Primary clauses modified	Description
11/09/09		LB	All	Merged sections from Google docs an formatted in the OGC template.
11/09/15		TO	PUCK section	Added OGC recommendation and additional figures.
11/09/16		LB	All	General edition.
11/09/17		CR	Section 6	General edition and updated recommendations.
11/09/18		ED	Section 9	Added IEEE SML harmonization section.
11/09/19		LB	All	General edition.
11/09/24		LB	All	General edition.
11/09/30		LB	All	General edition.
01/21/09	0.19 r2	LB	All	General edition.

1.5 Future work

Improvements in this document are desirable to amplify details of the specifications and resources used within the OGC standards. Future work can include topics discussed in section 2. Not all of them were addressed. The remaining topics can be subject for further work.

1.6 Normative References

The following documents are referenced in this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

OGC 07-036 OpenGIS Geography Markup Language (GML) Encoding Standard, 3.2.1, 2007-08-27, http://portal.opengeospatial.org/files/?artifact_id=20509.

OGC 07-022r1, Observations and Measurements – Part 1 - Observation schema 1.0, 2007-12-08 http://portal.opengeospatial.org/files/?artifact_id=22466.

OGC 04-094, Web Feature Service Implementation Specification, 1.1.0, 2005-05-03, http://www.opengeospatial.org/standards/wfs.

OGC® 07-000, OpenGIS® Sensor Model Language (SensorML) Implementation Specification, 1.0.0, 2007-07-17, http://portal.opengeospatial.org/files/index.php?artifact_id=21273&passcode=fxphjb8qrca4gwy7g626.

OGC 06-121r3 OGC Web Services Common Specification, http://portal.opengeospatial.org/files/?artifact_id=20040.

OGC 04-095, OpenGIS Filter Encoding Implementation Specification, 1.1.0, 2005-05-03, http://portal.opengeospatial.org/files/?artifact_id=8340.

1.7 Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Specification [OGC 06-121r3] clause 4 of Sensor Observation Service[OGC 06-009r6] and Clause 4 of Observations and Measurements – Part 1 [OGC 07-022r1].

1.8 Conventions

1.8.1 Abbreviated terms

API Application Programming Interface

GML Geography Markup Language

ISO International Organization for Standardization

OGC Open Geospatial Consortium

OWS OGC Web Services

OWL Web Ontology Language

O&M Observations and Measurements

MMI Marine Metadata Interoperability Project

SensorML Sensor Model Language

RDF Resource Description Framework

SAS Sensor Alert Service

SOS Sensor Observation Service

SPS Sensor Planning Service

SWE Sensor Web Enablement

TML Transducer Markup Language

UML Unified Modeling Language

XML eXtensible Markup Language

1.9 UML Notation

Some diagrams that appear in this specification are presented using the Unified Modeling Language (UML) static structure diagram, as described in Sub clause 5.2 of [OGC 06-121r3].

1.10 Background

The Southeastern Universities Research Association (SURA) hosted a workshop in Baltimore October 2005 called OOS Tech 2005 (note: OOS = Ocean Observing System). The workshop included approximately 100 ocean scientists, data mangers and computer science experts from around the country. They learned and talked about “Web Services for Interoperable Ocean Science.” After the workshop, a subset of the group agreed to work together on a follow-on activity to implement some of what they had learned. The agreed to build from their previous experiences using OGC WMS and WFS specifications. In previous years, they had built some basic elements of a Service Oriented Architecture (SOA) demo at OpenIOOS (www.openioos.org).

The OOS Tech 2005 follow-on activity began with 5 loosely defined goals: (1) Develop an end-to-end demonstration of web services increasing the interoperability of various regional real-time, ocean-observing programs, (2) gain experience with data exchange using SOAP with different tools on multiple platforms and implementations (3) leverage previous experiences with WMS and WFS, (4) leverage the Marine Metadata Interoperability demo focused on semantic interoperability using RDF-based ontologies, (5) leverage results of a NOAA Coastal Services Center salinity workshop in September 2005.

The small OOS Tech follow-on team formed their own “service-definition” team and began developing some simple SOAP interface definitions that leveraged various other OGC specifications, including GML, Observations & Measurements and SensorML. Since then the group has gain momentum and a project OOSTethtys got establish in 2006. OOSTethys members decided that working with standards organizations to pick the best standards, exercise them and advance them to bring observation system together,

was the logical path to move forward. OOSTethys members started an OGC Ocean Science Interoperability Experiment (Oceans IE) in 2007.

The timing of the OOSTethys and Oceans IE coincided with advances in the OGC Sensor Web Enablement (SWE) initiative. SWE capabilities prompted investigation between the WFS standard and the relatively new Sensor Observation Service (SOS). Extensive investigation, software development and real-world testing resulted in the set of open source SOS reference implementations and community cookbooks on OOSTethys.org.

The Oceans IE Phase I ended in May 2008, when the report was submitted. The Oceans IE Phase I investigate the use of OGC Web Feature Services (WFS) and OGC Sensor Observation Services (SOS) for representing and exchanging point data records from fixed in-situ marine platforms. The Oceans IE Phase I produced an engineering best practices report and reference implementations for using OGC Sensor Observation Service. The best practices from this experiment make possible the consideration and adoption of SOS by NSF's Ocean Observing Initiative, the U.S. government Integrated Ocean Observing System, Data Integration Framework, and Europe's ESONet program.

Due to the interest of the OOSTethys participants in Phase I, they decided to engage on a Phase II advancing topics of interest that were depicted as future work in the phase I. A summary report for each working group is discussed in the following sections.

2. Working groups and activities

OSIE II kickoff date was on March 27, 2009. A one hour meeting was held every week to track progress, prioritize and discuss main issues. All the minutes are available at the OOSTethys web site. OSIE II is planned to end on November 2009 after this report is submitted to OGC.

Working groups were created based on interest of the participants. The common theme was Sensor Web Enablement for ocean data. The following 21 topics summarize the collective interests:

1. **Automated metadata/software installation via PUCK protocol.**
2. SWE Common encoding for vertical and horizontal profiles (e.g. ADCP) and trajectories (AUV).
3. Long time series services (e.g. 20 years of data).
4. **Offering of complex systems (e.g. observations systems containing other systems) such as collection of stations.**
5. **Linking data from SOS to out-of-band offerings.**
6. Representation of vectors and scalars in SOS vs semantics.
7. **Semantic Registry and Services.**
8. Alert services for fast detection of coastal events. Offerings that are event based (e.g. all tsunami sensors within +/-12 hrs of an tsunami).
9. XSLT and SOS responses.
10. **CSWRegistry.**
11. Portal - Human Interface to discover and download access data
12. Development of KML encodings for SOS.
13. WCS / SOS Chaining

14. NetCDF/OpenDAP and SOS Chaining Gridded data and stations.
15. **IEEE-1451/OGC-SWE harmonization**
16. Instruments control.
17. SOS WaterML harmonization
18. Incorporation of QA/QC into SOS services.
19. Deployment of SOS services
20. SOS client – visualization
21. Guidance for capturing metadata fields and lineage using SensorML.

Only the bold topics were advanced enough to be presented in this report. For each selected topic, the goals, motivation, list of participants, summary of the work and recommendations and conclusions will be further explained.

3. Topic: Automated metadata/software installation via PUCK protocol

3.1 Goals

This team had the following goals:

- Demonstrate automated retrieval and installation of IEEE 1451 and OGC SWE components from instruments that implement MBARI PUCK protocol. These components included IEEE 1451 TEDS, SensorML documents, and instrument driver software to be executed on the instrument "host" computer.
- Experiment with approaches to automatically detect when a sensor has been installed, removed, or exchanged.

3.2 Motivation

Standards such as OGC SWE and IEEE 1451 strive to integrate diverse instruments into networks with minimal human effort and high reliability. Use of these standards requires several software components that must be installed on the instrument network, including instrument "drivers", web servers, and metadata documents that describe instruments in a standard way. Most instrument networks today require careful manual installation and configuration by technicians to assure that the software components are properly associated with the physical instruments that they represent. In oceanographic applications, these installation and configuration steps often must be performed in shipboard environments that are physiologically and psychologically challenging, thus increasing the possibility of human procedural errors.

MBARI PUCK addresses these installation and configuration challenges by defining a standard instrument protocol to store and automatically retrieve metadata and other information from the instrument device itself. This information can include OGC SWE SensorML and IEEE 1451 TEDS documents, as well as actual instrument “driver” code. A host computer that understands PUCK protocol can automatically retrieve and utilize this information from the instrument itself when the device is installed. For example, components required by OGC SWE and IEEE 1451 can be physically stored with instruments and sensors and automatically installed on a sensor network, when the instrument is plugged in, thereby eliminating tedious and error-prone manual configuration steps.

MBARI PUCK defines a small standard “PUCK datasheet” that can be retrieved from a PUCK-compliant instrument. The datasheet includes a universally unique identifier (UUID) that is guaranteed to be unique among all PUCK-enabled instruments, as well as manufacturer and model codes. All compliant instruments must supply the datasheet. In addition MBARI PUCK defines an optional “PUCK payload” that contains additional information needed to operate the instrument; this can include instrument driver code and metadata. MBARI PUCK does not limit the payload format or content, leaving that decision up to observatory developers and users.

MBARI PUCK protocol augments but does not replace existing instrument protocols. Thus a manufacturer can modify their instrument’s firmware by adding PUCK commands to the already existing command set of the instrument. This approach allows manufacturers to implement MBARI PUCK without abandoning their existing firmware and software applications.

The majority of today’s oceanographic instruments have a serial RS-232 interface, compatible with underwater low-power applications. MBARI PUCK protocol is intended to be a software protocol, compatible with most existing physical instrument interfaces. Thus MBARI PUCK v1.3 is applicable to RS-232 instruments, and uses just the RX, TX, and GND signals.

3.3 Participants

Participants on this topic included:

- Polytechnical University of Catalunya (UPC-SARTI) – Joaquin del Rio Fernandez, Dan Toma
- Christian Albrechts University at Kiel – Jesper Zedlitz
- Bremen University – Christoph Waldmann
- SmartBay Canada – Neil Cater, Eric Davis
- Axys Technologies – Chris Ng, Reo Phillips
- Compusult Ltd – Angela Amerault, Robert Thomas
- RBR Ltd – Greg Johnson, Graham Jones
- SEND Offshore Electornics GmbH – Klaus Schleisiek
- MBARI – Kent Headley, Carlos Rueda, Tom O’Reilly

3.4 Discussion

3.4.1 Automatic installation of IEEE-1451 and OGC SWE components using instruments that implement MBARI PUCK protocol

Figure PUCK-1 shows the basic system architecture of the PUCK-1451-SWE test-beds developed and demonstrated for this project. At the lowest level are RS-232 instruments that implement MBARI PUCK protocol in addition to their manufacturer-specific protocols. These are plugged into serial ports on an observatory node, which are located

at UPC-SARTI, Kiel, and MBARI. Each observatory node presents an IEEE 1451.0 HTTP interface to the Internet.

Engineers at each observatory developed instrument drivers and other components suited to their particular observatory infrastructure. While these components were implemented differently for each observatory, they all utilize OGC SWE SensorML and IEEE 1451 TEDS, and support IEEE 1451.0 protocol. Technicians used MBARI's PUCK utilities to store instrument-specific components – drivers and metadata – in the PUCK-enabled instruments. Now when the instruments are physically plugged into observatory node serial ports, the node can retrieve and install the components, using PUCK protocol.

Network clients can access the observatories and instruments through IEEE 1451.0 protocol. One such client is the Smart Transducer Web Service (STWS) developed at the National Institute of Standards and Technology, which was designed to map between IEEE 1451 and OGC SWE protocols [Song and Lee, 2007]. The STWS in turn presents the proposed standard IEEE 1451 STWS interface to the network, which can be accessed by clients such as the STWS-SOS developed by Northrup Grumman. The SOS provides instrument access to OGC-SWE clients through SOS protocol. PUCK protocol supports these clients in two ways:

a) The TEDS and SensorML documents retrieved by clients originates within the instruments themselves, and are initially retrieved by the observatory node through PUCK protocol.

b) The instrument drivers executing on the observatory node support and implements element of the IEEE 1451 protocol, thus enabling access by clients. The observatory node retrieves the driver code from the instrument using PUCK protocol, and then executes that code.

This architecture was described by [O'Reilly, Headley et al 2009] and demonstrated at various stages of development at several venues, including the Ocean Innovations 2008 Interoperability Workshop, the NSF OOI Sensor Workshop, and the Third International Workshop on Marine Technology.

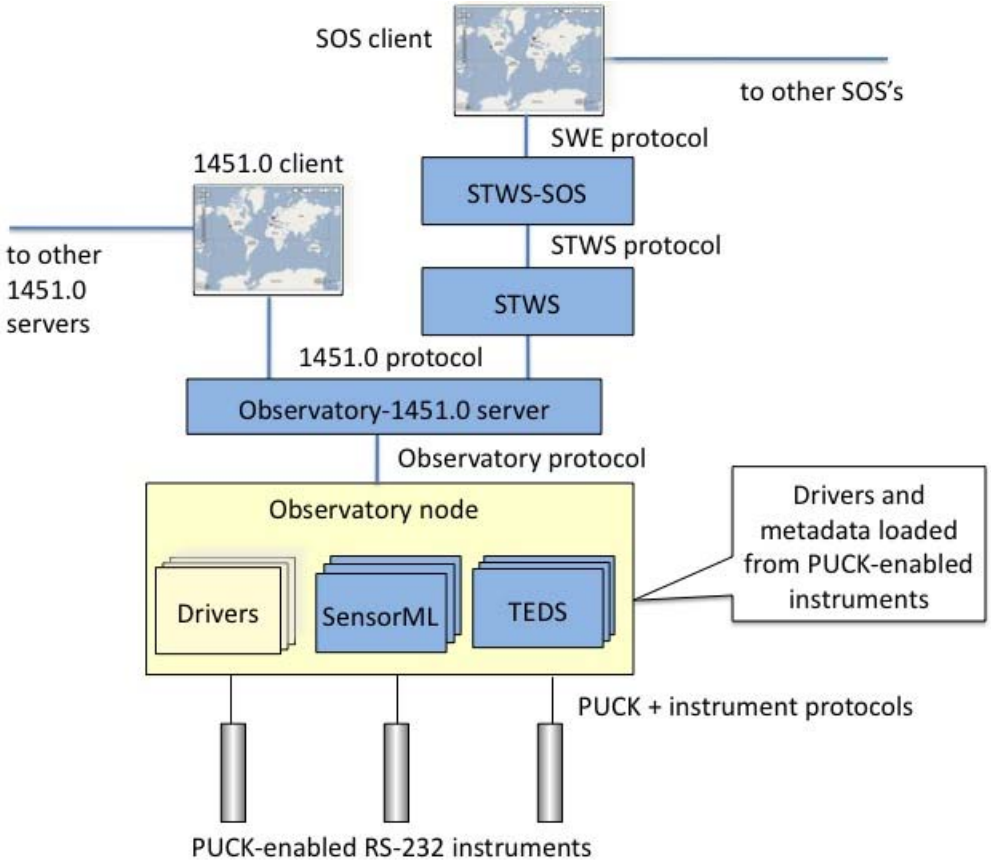


Figure PUCK-1: Basic architecture of PUCK-1451-SWE test-bed

In addition to this basic architecture, Compusult developed an “adapter” approach that maps directly between Sensor Observation Service protocol and observatory protocol, thus by-passing IEEE 1451.0 protocol. Compusult and MBARI teams collaborated to demonstrate how MBARI’s “SIAM” instrument middleware could be directly accessed with a Sensor Observation Service through a Compusult-developed SIAM-SOS adapter. SIAM was developed before the advent of Sensor Web Enablement, and is portable to low-bandwidth systems for which Web Service HTTP protocols are not suited. The SIAM instrument service interface provides Java RMI methods to configure the instrument, retrieve static and dynamic metadata, and of course acquire data. [O’Reilly, Headley et al 2006]. Many of these methods have a straightforward logical mapping to Sensor Observation Service operations. For example, SIAM's Instrument.getMetadata() provides a standardized instrument description document, corresponding to the SOS DescribeSensor operation. SIAM's Instrument.acquireSample() and Instrument.getPackets() method are analogous to the SOS GetObservation operation.

The Compusult team incorporated this logical mapping into an adapter component that translates between SOS and SIAM protocols. Thus the SOS can be readily integrated with the "legacy" SIAM instrument service. Figure PUCK-2 shows this architecture, and Figure PUCK-3 describes the sequence of events when retrieving a SensorML document from a PUCK-enabled sensor.

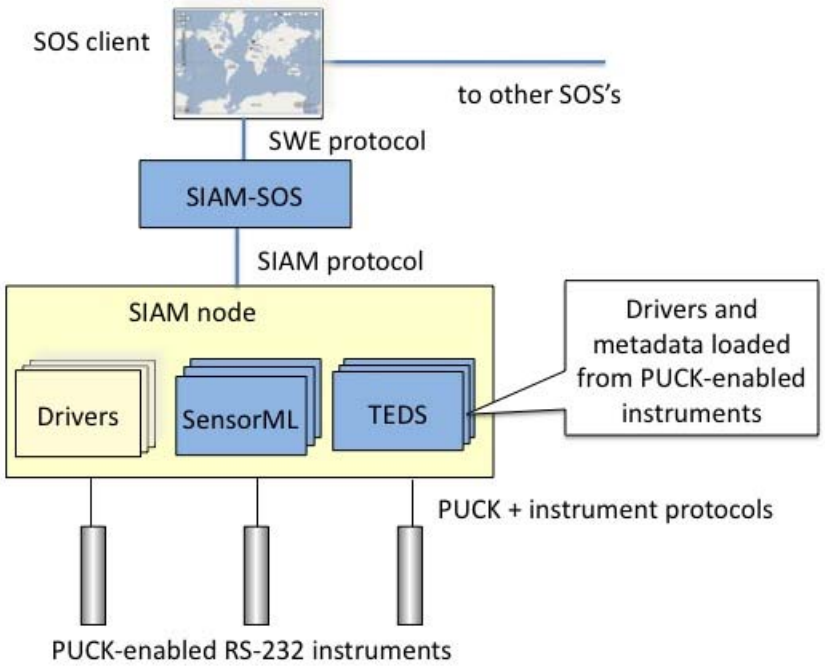


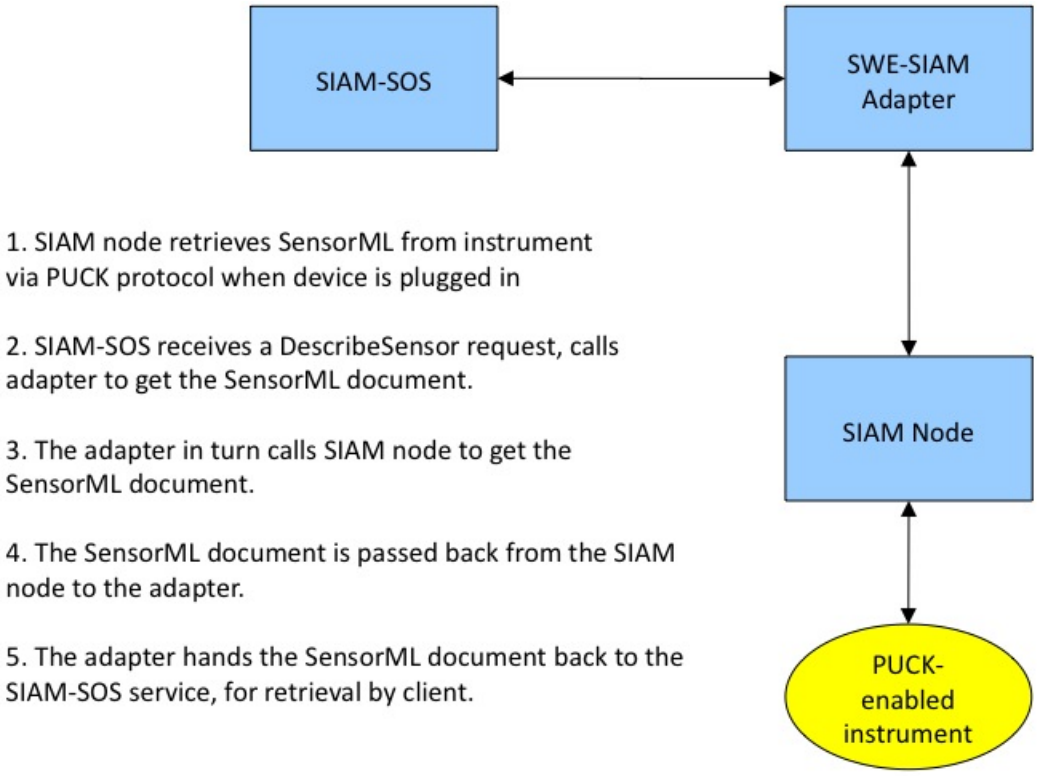
Figure PUCK-2: Integration of Compusult SOS with MBARI-SIAM

This direct observatory-SOS adapter approach requires fewer components than the basic PUCK-1451-SOS architecture described earlier, and is simpler. Obviously the observatory cannot be accessed through IEEE 1451 without a 1451.0 web server.

Figure PUCK-3: Sequence describing retrieval of SensorML via PUCK protocol, SIAM, and SOS DescribeSensor operation.

3.5 Multiple PUCK payload components

In the previous section we described how instrument SensorML and TEDS documents as well as driver code can be stored in an instrument's PUCK payload. Note that the instrument driver utilized in a particular observatory may not be applicable to another observatory. In other words, observatories may conform to standard network interfaces and metadata formats, but implementation details will differ from one observatory to the next depending on computing hardware, communications infrastructure, development history, and other factors. A particular observatory may also require non-standard configuration information for each instrument.



Nonetheless, we demonstrated how a single instrument can contain components for several different observatories in its PUCK payload and how each observatory can search the payload for the components that it needs. Thus a single PUCK-enabled instrument can be plugged into various observatories, and be automatically integrated into each. For example, the UPC-SARTI OBSEA test-bed retrieves and utilizes a IEEE 1451 TEDS XML document from the instrument. MBARI observatories retrieve and utilize a jar file containing MBARI SIAM instrument service code and metadata. We demonstrated how both of these payloads can be stored simultaneously in instrument payload memory, with each payload preceded by an identifying XML tag, as shown in Figure PUCK-4. The tag specifies the name, type, and checksum of the payload, as well as the PUCK memory byte address of the next payload tag. Each tag is immediately followed by the actual payload component. Thus a particular observatory host can efficiently read through an instrument's PUCK payload until it finds a component of the desired type. Based on our experiments we propose an addendum to the PUCK v1.3 specification that defines a standard layout of tags and payloads, standard tag labels and payload component type names. This standardization will enable easy sharing of PUCK-enabled instruments between observatories with different architectures.

In addition, the Kiel team has demonstrated the use of digital signatures on payload components, which the host computer can examine to verify that the components come from a trusted source. This type of security verification could become especially important as instruments are exchanged between observatories.

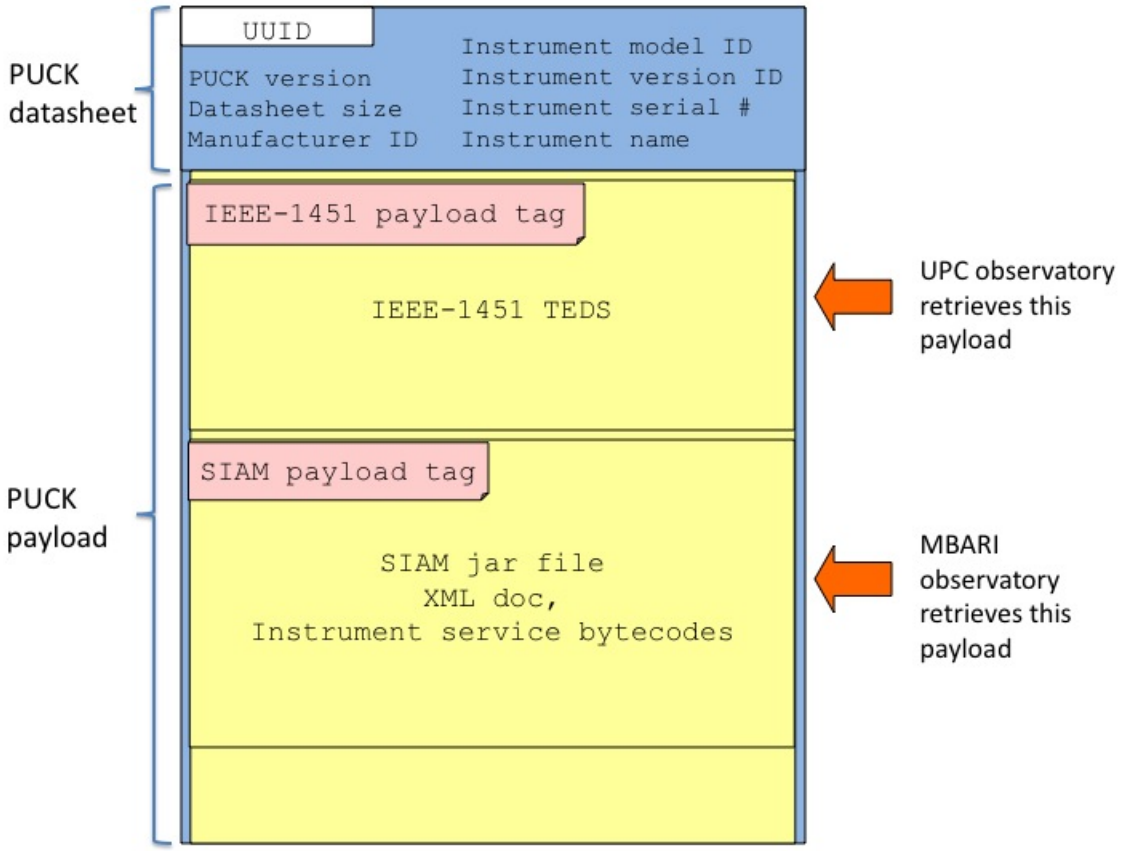


Figure PUCK-4: PUCK memory map example showing multiple “tagged” payload components

3.6 Instrument installation and removal detection

As noted earlier, MBARI PUCK protocol requires just RX, TX, and GND RS-232 signals in order to be compatible with existing oceanographic instruments and applications, connectors and cables. Oceanographic instruments are often deployed on the end of long cables, e.g. hanging from a mooring. Many oceanographic instruments are also deployed on platforms that are limited in available power. The RS-232 serial protocol is compatible with these constraints, and so is the most common oceanographic instrument interface. In addition, underwater systems are usually designed to minimize the number of wires in order to control housing and connector complexity, cable weight, and cost.

Thus unlike USB or IEEE-1451.2, PUCK protocol does not utilize a single dedicated connector pin signal to detect when an instrument is physically installed or removed from a host computer port. Instead other approaches that utilize just RX, TX, and GND must be used to determine when these events have occurred. The OSIE-PUCK teams investigated several approaches to detect instrument installation and removal:

a) PUCK detection at boot time: In this approach, the host computer attempts to contact instruments with the PUCK "soft break" command on each serial port immediately after the host is booted. The soft break must be issued at all possible baud rates since PUCK protocol does not specify a "discovery" baud rate. If the host receives a PUCK response from a port, it can then retrieve the PUCK datasheet and optional payload from the

instrument and utilize them to load the appropriate instrument driver and metadata. If a PUCK response is not received at any baud, then the next serial port is tried until are ports are checked. This approach was implemented on an Axys Technologies Watchman500 buoy controller by the SmartBay and Axys Technologies teams. In this case, the controller utilizes the PUCK datasheet's instrument manufacturer and model codes to load an appropriate driver from a library stored onboard the controller. After all ports have been processed, the Watchman500 intializes all discovered instruments and goes into normal operations mode. This approach requires the instrument host computer to be rebooted when instruments are installed or removed. However this requirement is quite acceptable for many systems in which instruments are changed relatively infrequently.

b) Manual notification of instrument installation and removal: In this approach, a human operator runs a simple utility that notifies the instrument host that a PUCK-enabled instrument has been installed or removed from a serial port. When notified that an instrument has been physically installed, the host uses PUCK protocol to automatically retrieve the PUCK datasheet and optional payload, and installs appropriate instrument drivers and metadata. This approach sacrifices automated device detection and "hot-swapping" but conserves power and avoids safety and corrosion issues associated with applying power to exposed underwater wires. MBARI currently uses this approach on its deployed buoy-based and cable-to-shore observatories.

c) Instrument detection based on serial port file existence and asynchronous data detection: The Kiel team's instrument host utilized USB ports and USB-to-serial adapters to communicate with serial instruments, and the team developed a simple algorithm to detect instrument plug-in based on the existence of the USB serial port. For actual serial ports, their host software automatically detects the presence of streaming instruments by the asynchronous arrival of data at the serial port. The latter technique is limited to streaming instruments, i.e. it does not apply to synchronously polled devices. Once an instrument is detected, PUCK protocol was utilized to retrieve the instrument's metadata and a jar file containing the driver code.

d) Automated detection of installation installation and removal using PUCK protocol: The UPC-SARTI team developed a "hot swapping" approach that does not require any manual steps other than physical installation or removal of an instrument. Figure PUCK-5 illustrates this algorithm as a flowchart. The host computer periodically interrogates the serial port for a PUCK-enabled instrument by issuing a PUCK "soft break" command. If the host receives a PUCK response from the serial port, the host retrieves the 96-byte PUCK datasheet and examines the UUID to determine if a new instrument has been installed (the UUID is guaranteed unique to each instrument). If so, the host retrieves the SensorML and IEEE 1451 TEDS description from the instrument's PUCK payload, loads an appropriate driver and configures the newly detected instrument. Finally the driver begins retrieving data samples from the instrument at some interval I_{SAMPLE}. If I_{SAMPLE} is greater than the time needed to query an instrument for its PUCK datasheet ($T_{PUCK-CHECK}$), then the instrument driver will attempt to read the PUCK datasheet before each sample, thus detecting when the instrument has been removed or replaced with another. If on the other hand I_{SAMPLE} is shorter than $T_{PUCK-CHECK}$, then the host checks the serial port for a PUCK response only if an error is encountered when attempting to communicate

with the instrument. This algorithm presumes that replacing a fast-sampling instrument with another will always result in a communications error. However note that if the instrument is quickly replaced with another of the same model, a communications error might not occur and hence the host would not be aware that the instrument was replaced. Thus subsequent data samples would not be associated with the correct instrument and metadata. Therefore users must be aware that when swapping fast-sampling instruments of the same make and model they should leave the instrument port empty for at least I_{SAMPLE} to ensure that the algorithm will properly detect the new instrument.

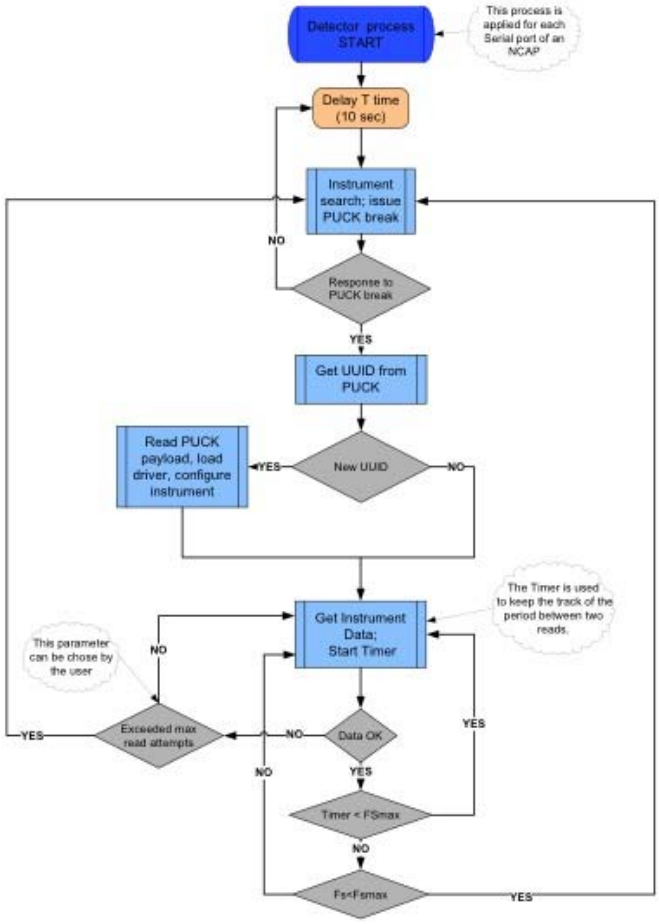


Figure PUCK-5: Automated instrument detection algorithm developed at UPC-SARTI .

3.7 Effort required to integrate PUCK

Teams report a variety of effort required to integrate PUCK protocol into their observatory test-beds, using a variety of approaches:

SmartBay-Axys - Modified Axys Watchman500 buoy controller to automatically detect PUCK-enabled instruments and utilize PUCK datasheet to load appropriate instrument

drivers from onboard library when controller reboots. Required effort: approximately 10 engineering days, including testing and interfacing with sensors to ensure correct and seamless operation. Beyond this, support is currently being provided for the integration of the first operational buoy.

Christian Albrechts University at Kiel - Detect instrument plug-in, retrieve, verify, and utilize driver code from PUCK payload. Required effort: approximately 3 engineering days.

UPC-SARTI Vilanova - Automatically detect instrument installation, retrieve IEEE 1451 TEDS and SensorML from instrument, instantiate and configure instrument driver accordingly. Make data accessible to IEEE-1451 server as well as SWE Sensor Observation Service. Utilized payload tags for instrument interoperability between observatories. Required effort: approximately 9 engineering days.

Compusult Ltd: Developed SOS-SIAM "adapter" component that maps between Sensor Observation Service protocol and MBARI's SIAM middleware protocol (SIAM service code and SensorML installed via PUCK protocol). Required effort: about 7 days.

3.8 Recommendations

The OSIE-2 PUCK project has resulted in a few suggested modifications to MBARI PUCK protocol v1.3, including the following:

1. Standardized PUCK payload tags: As described above, we investigated storage of multiple payloads components within a single instrument, thus enabling that instrument to be used within several different observatory architectures. We recommend that the PUCK v1.3 specification be amended to define standard tag format, elements, and type names.

2. Relaxation of PUCK protocol timing requirements: the current PUCK v1.3 specification is rather rigid with respect to timing requirements of the PUCK "soft break" command and timeouts for other PUCK protocol commands. The team's experience with instruments accessed across a network through serial-to-Ethernet adapters indicates that these timing requirements should be relaxed to some extent. E.g. the document currently specifies a 750 millisecond pause between PUCK soft break components; this could be relaxed to something like "between 750 millseconds and 1 second". Further tests are needed to determine reasonable bounds on these timing requirements.

We also have some recommendations regarding future OGC SWE experiments and implementation approaches:

a) Automatic instrument registration: MBARI PUCK now provides an automated method to install SensorML onto an instrument host, for distribution to the broader sensor network. A logical next step is to develop and refine techniques for automated registration of the instrument with a catalog service immediately after the SensorML has been retrieved from the instrument.

b) Storage and retrieval of geolocation data: The SensorML describing an instrument and its capabilities can now be stored and retrieved from the instrument itself. Ideally, we believe that instrument manufacturers should deliver the SensorML with the instrument. SensorML can encode instrument geolocation, and clients commonly depend on this. However in many cases the instrument manufacturer will not know where the instrument will be used, and so the geolocation must be omitted from the manufacturer-provided SensorML. In other cases the user may use the instrument for a while in one location, then move it – thus someone must update the instrument's PUCK payload with the new location, if the system relies on the SensorML's geolocation element. Perhaps an observatory software component could determine location from GPS or other means, and dynamically insert location into the SensorML when requested by a client. Alternatively, clients could rely on geolocation observation data rather than SensorML. We recommend that various approaches be explored further.

4. Topic: Linking data from SOS to out-of-band offerings.

4.1 Goals

This team had the following goals:

- Advance the understanding and intent of the *out-of-band* mode in SOS
- Compare existing distributed systems for real time (RT) data distribution

4.2 Motivation

The OGC SOS specification talks about an "out-of-band" option, which has not been implemented in the majority of SOS Services. For example, the oceanographic community was not aware of an SOS implementing using the out-of-band option. The precise intent of the out-of-band option is not very clear. For example, it can be an SOS server offering a file that can be downloaded form a server or a connection to a real-time message queue endpoint.

The team for this topic was concerned about performance characteristics when integrating distributed systems. These characteristic include Quality of Service (QoS), frequency, and asynchronous push capabilities. Components in this system integration include OGC SOS, MBARI PUCK, Unidata IDD, NSF DataTurbine and OMG Data Distribution Service (DDS). There is a benefit if implementers can have the ability to share common practices and identify gaps in the OGC SOS specification in regard to common issues such as how to specify and deliver real-time qualities of service.

4.3 Participants

- Dave Coyle (USGS)
- Ben Domenico (UNIDATA)
- Sameer Tilak (UCSD)

4.4 Discussion

4.4.1 SOS XML and REST

The OGC has defined the SOS protocol and related schemas in terms of primarily XML and Web technologies. While it is understood that the core model is a conceptual model (in fact UML is used in the specification), the de-facto implementation has been based on XML and the W3C XML Schema Language (XSDL). As a result of this evolution, implementers tend to make the assumption that SOS must be implemented using XML. This is an important system design consideration when planning to integrate SOS into a scalable distributed system with "near real-time" capabilities. The OGC has addressed this concern in part by bringing forward efficient XML technologies; however, efficient XML does not provide all of the technical answers. The key remaining issues are: how to provide content-based routing; how to provide near real-time qualities of service; how to implement the core, edge, and end-point integrations.

The OGC protocols were originally based on IETF POST. The OOSTethys group and others have now tied the SOS protocol to IETF HTTP-1.1 GET. This is also known as REST web services. HTTP POST and GET are synchronous request/response protocols. This fact brings forward the folowing questions: How does REST tie in to a real-time distributed system? Is it possible to use SOS over REST and yet somehow achieve asynchronous notification services? Is SOS REST suitable as a gateway into a real-time asynchronous notification based system? If so what is missing from the SOS REST protocol? Even if we use a SOS/REST adapter (e.g. using the Atom Publishing Protocol or polling/token), how does this integrate with the core distributed system architecture (e.g. a message-queue based system).

In the classic SOS/REST style web-service, the HTTP Client is always the initiator (recall that HTTP is strictly a request/response protocol). If the OGC SOS specification is tied to HTTP GET then by definition the system is limited by this request/response message exchange pattern. The same case is true for HTTP POST from the original OGC specification. So the problem is how can implementers of SOS services go beyond the limits of SOS/REST to achieve a push model. A related question is how do we realize near real time "qualities of service", such as reliability, durability, frequency, and liveliness. The answer is that we must derive these factors from existing real-time distributed systems. Some of the systems considered in this report are: OMG DDS, Unidata IDD, and NSF DataTurbine.

OGC has defined specification and protocols such as Transducer ML; however, these efforts have seen low levels of activity and interest. As mentioned above, the answer lies in building effective integrations with existing specifications such as the OMG DDS, or with existing systems such as the Unidata IDD. Another interesting integration along these lines is the IEEE 1451 PUCK protocol , discussed in the previous section.

Should there be a core SOS specification for RT-SOS that is "protocol agnostic"? The answer is most likely negative in this case. That said, each implementor should attempt to share with OOSTethys and other groups as he or she integrates the SOS model with the target system's QoS and Routing features. By sharing ideas and building a community going forward, we can discover the key use-cases for RT-SOS. Should new SOS

asynchronous streaming message types be added? In the original SOS Spec HTTP POST was used to define messages. It will be important to ensure that the REST mapping does not impose a request-response bias on the SOS protocol. A clearly defined streaming SOS protocol is needed.

It is possible that SOS asynchronous streaming message types should be added. Once this is done system designers can use the SOS message types to build non-HTTP based solutions. Some of the solutions we have discussed are: Unidata IDD , OOI-CI , NSF DataTurbine, and OMG DDS. Each of these systems has similarities and differences; however, the question is the same: what is the unifying data model for building OGC SOS distributed systems?

As mentioned the question originated in the course of discussions in the OOSTethys group, in which it was speculated that the SOS "out-of-band" feature of the OGC Sensor Observation Service implies a sort of escape mechanism from the restrictions of HTTP GET. This distinction may refer to the HTTP return channel, or perhaps it refers to the OGC/O&M "result" mechanism, or the HTTP Content-Type. It may even refer to the request/reply message exchange pattern; at least the discussion brought this question forward.

4.4.2 HTTP GET

There are several conflated issues here. These issues must be cleared before we can continue with question of real-time and push RT-SOS services. The first issue to address is the definition of HTTP GET or "REST web services" (which is evolving as a common practice). In terms of SOS as an HTTP GET (REST) web service, what is the meaning of the OGC "out-of-band" property? The following issues should be addressed in this context:

- Does this refer to the OGC/O&M "result" mechanism? If so what is the definition of this concept?
- Is this the web version of the classic "protocol bootstrap mechanism"; i.e., similar to JINI?
- Is this different from HTTP GET and IANA MIME/Content-Types? If so, is this in opposition to emerging REST recommended practices?

The IETF HTTP-1.1 standard specified that the (SOS) HTTP GET response is requested as a IANA MIME Media Type, and that the correct Content-Type is returned according to the specified matching rules. Therefore in principle the SOS out-of-band cannot be meaningful in this context, unless it is simply an "indicator tag" that is used in the OGC SOS GetCapabilities document.

In terms of the Media Type and returned Content-Type, it is suggested that OOSTethys and other SOS implementers should register one or more new IANA Mime-Types. The first Mime-Type might specify the XML SOS response. In terms of RT-SOS, perhaps there is justification for additional Content-Types that defines the stream-based protocol. This may be stating the obvious but perhaps it should be brought forward for more

focused discussions. This change would address the required questions about content-encoding and security. In other words, content negotiation is done by the HTTP-1.1 layer.

4.4.3 Real Time

The question of HTTP GET (REST) is an inevitable distraction to the more important core question how to build effective real-time solutions. The answer is that both channels are needed. HTTP GET may have a role to play in service discovery or possibly as a gateway or protocol bootstrap mechanism. Implementors should consider internal system goals but also OGC interoperability. The choice of the core software for data distribution is contingent on the deployment environment. Within these core architectures, implementers may need to encode the SOS/O&M content models in more efficient forms such as IDL, ASN.1 PER encodings, or a combination of encodings (e.g., some string metadata should be retained for real-time content-based routing purposes). The type of network deployment is a critical decision. The OMG DDS offers a flexible software architecture that can be used in any network deployment. This flexibility may provide value for the organization building the given SOS-capable distributed system. Quality of Service (QoS) and Service Level Agreement (SLA) are terms that apply to distributed systems. These QoS terms are absent from the OGC SOS standard specification. There may be a good case to be made for some real-time QoS properties to be added to the OGC SOS standard, for example in the GetCapabilities document and in the GetObservations response specification. Quality of Service (QoS) is the key term that applies when the goal is to build a so-called "real-time" or "near-real-time" Sensor Observation Service (SOS). The SOS can be a stand-alone solution or a gateway into an existing solution. The Unidata IDD is an example of a near-real-time service that is actually a gateway into an existing real-time deployed solution.

4.4.4 Network Deployment Scales

In all cases it is important to understand the separation of concerns in at least three scales of network deployment implementation:
1. On Wide Area Network (WAN) deployments it is doubtful that SLAs and effective QoS can be realized. In principle it is possible on traditional carrier scale networks such as ATM (Asynchronous Transfer Mode) over fiber optic fabric with LANs as edge-devices. For example, ATM has the ability to do "qos reservations". The point is that it depends on the network.
2. QoS is possible on the Local Area Network (LAN). For example, the OMG DDS is being used on specialized LAN networks such as in financial trading systems, ship-board systems, and in other military and industrial applications.
3. QoS is certainly possible in the case of Embedded Systems. In practice the data will be internally handled as an alternate encoding. Of course this may be where PUCK and similar solutions enter into the picture. It may be that embedded systems have no relationship with OGC SOS beyond the fact that it is an integration problem.

4.5 Comparison of Distributed Systems

As described the previous section, implementers of distributed and embedded systems that interface with SOS should try to ensure they provide sufficient SOS data and meta-data artifacts. In most cases this integration must be performed manually. In some OMG DDS implementations, the SOS and OGC Observations and Measurements data models can be automatically mapped to efficient and routable IDL data structures. This approach ensures both compliance and scalability. This section provides details about some of the systems considered and for which OGC SOS integration work may be in progress.

4.5.1 UniData IDD and LDM

Implementers who prefer to use an existing system may have some good choices. It is possible to integrate the existing Unidata IDD system with an SOS gateway or "edge-service" in the form of an SOS HTTP GET web service endpoint. This is done by implementing an IDD/LDM (Local Data Manager). Since Unidata IDD is in fact an existing globally deployed and scalable real-time system – this could be an excellent option if it is practical to do so. For example, Tony Cook (TAMU) has developed a working integration between IDD/LDM and SOS.

The Unidata community of over 400 university departments is building a system for disseminating near real-time earth observations via the Internet. Unlike other systems, which are based on data centers where the information can be accessed, the Unidata IDD is designed so a university can request that certain data sets be delivered to computers at their site as soon as they are available from the observing system. The IDD system also allows any site with access to specialized observations to inject the dataset into the IDD for delivery to other interested sites.

The Unidata Local Data Manager (LDM) is a collection of cooperating programs that select, capture, manage, and distribute arbitrary data products. The system is designed for event-driven data distribution, and is currently used in the Unidata Internet Data Distribution (IDD) project. The LDM system includes network client and server programs and their shared protocols. An important characteristic of the LDM is its support for flexible, site-specific configuration.

4.5.2 OOI-CI

OOI CI is incorporating much of the design requirements of IDD but the underlying push mechanism, the OOI Messaging Service, is built on a newer generation of Inter-Process Communication (IPC) technologies that are based on set of open standards, AMQP (reliable asynchronous multi-party messaging), FIPA (extensive message header specification that spans conversation, semantic, syntactical and encoding tagging) and ASN1/PER (encoding rules for packed binary transfers - this one is still under investigation - we have reviewed a good number of specs).

The OOI IPC infrastructure can be tuned for a wide range of communication environments; from intermittent satellite communications, to standard public internet, to high bandwidth wide are lambda circuits, to RDMA and to internal shared-memory multi-core processing systems.

4.5.3 GTS and NOAAPORT

The Unidata IDD provides GTS data which can be accessed via a local LDM adapter implementation. Several organizations are interested in accessing IDD data products on the GTS. Unfortunately funding is not always available to support these implementations. For example, the IDD contains all of the GTS data that the NWS considers necessary for its operational use. This is not the entire GTS data stream, but it is quite a bit of it. You can check the contents of the IDS|DDPLUS data stream available in the NOAAPORT broadcast/IDD. Here are some pointers of interest.

- http://weather.unisys.com/noaaport/NOAAPORT_Channel_Content.html
- http://www.nws.noaa.gov/om/marine/noaaport.htm
- http://www.nws.noaa.gov/om/marine/home.htm#observations
- http://www.unidata.ucar.edu/support/help/MailArchives/idd/msg04361.html

4.5.4 DataTurbine

The DataTurbine software emerged as a commercial product in the 1990s from collaborations between NASA and private industry. In October 2007, a grant from the USA National Science Foundation (NSF) Office of Cyberinfrastructure allowed the developers to transition DataTurbine from a proprietary software product into the NSF Open Source DataTurbine Initiative (http://www.dataturbine.org).

DataTurbine satisfies a core set of critical infrastructure requirements that are common across a number of observing systems initiatives, including reliable data transport, the promotion of sensors and sensor streams to first-class objects, a framework for the integration of heterogeneous instruments, and a comprehensive suite of services for data management, routing, synchronization, monitoring, and geo-spatial data visualization. It is an open source streaming data middleware, released under Apache V2 license. This allows implementers to get access to stream handles and execute functions directly on streams.

Since the OSDT product is written in Java, DataTurbine is highly portable and is available for many platforms, from 64-bit multi-core machines and desktop systems to handheld and embedded devices. As a concrete example, performance of DataTurbine was tested on a 8 core Sun Fire T2000 Server (16 GB memory, runs Solaris OS and is connected to a 9 TB storage (RAID), Dual-core Linux servers, to Gumstix devices, and cell phones.

The OSDT has support for in-network processing and time synchronization. It also has a support for Spatial Data and Visualization Services. For example, Google Earth integration with DataTurbine can be used to visualize real-time sensor data. The OSDT supports coupling sensor data with modeling tools. For example, it is possible to meld the rich image and numeric toolkits of Matlab with the real-time data streams from the DataTurbine. Matlab support exists out of the box.

The OSDT has been deployed in a variety of real-world streaming data applications such as coral reef monitoring, lake monitoring, animal tracking, airborne environmental

monitoring, animal tracking, earthquake engineering, and environmental sustainability to name a few.

4.5.5 OMG Data Distribution Service

The OMG Data Distribution Service (DDS) is a real-time middleware software component. The OMG DDS is similar in concept to the NSF DataTurbine product which is also described in this document. Similar to the NSF DataTurbine, the OMG DDS is a "real-time" middleware software component. The OMG DDS is based on the DDS standard and the DDSI protocol specification.

There is no existing deployment of the OMG DDS for the purpose of a real-time RT-SOS system. Such a system could be built using one of the commercial or open-source implementations. The proposed OMG DDS based distributed system could integrate with the OGC SOS protocol. This section provides some details about how this integration could be done. The value proposition for the OMG DDS is explained in some detail. The OMGS DDS could be used as the core component in an advanced real-time capable Sensor Observations Service "out-of-band" asynchronous systems architecture. Most of the solution architectures we will evaluate do not directly address the question of real-time in terms of issues such as quality of service. When they do address these concerns, the implementation is proprietary or the solution is not fully specified. The OMG Data Distribution Service (DDS) addresses these concerns.

The OMG DDS is a fully specified system. That is, the system definition is fully specified both in terms of software architecture and in terms of the wire protocol. The later is important for for vendor interoperability and for security assurance.

In addition the open-source implementation there are two principle vendors. The PrismTech OpenSlice product is an open-source commercial DDS product. The RealTime Innovations (RTI) RTIDDS product is a closed source commercial DDS product.

OMG DDS can be a solution for the near-real-time SOS requirement. A lighter weight and possibly more constructive approach might be to take in the lessons learned from a system such as the OMG DDS which has been designed by experienced engineers and developed within highly evolved and tested industrial quality products.

The OMG DDS provides the choice of a low-level API or a higher-level API which appears to the user as a real-time database system. Most developers use the lower level API. The higher-level API is called the DLRL, or Data Local Reconstruction Layer. Based on the DLRL, the OMG DDS can also be used by application developers as the equivalent of a real-time object-oriented database. The DDS can also be used as the data distribution solution in combination with a Postgres, MySql, or other database (i.e. based on triggers).

The OMG DDS could serve as an OOSTethys-endorsed best-practice solution for near-real-time data-consumer endpoints on a WAN-based network, or as a real-time solution in suitable closed environment such as on ship-board network or data-center network system. The OMG DDS solution has a wide range of applicability and will provide the

ability to build systems that are reliable, scalable, near-real-time or real-time in terms of quality of service.

The OMG DDS is extremely scalable due to its defined DDSI protocol and its peer-to-peer network architecture. The implementer can operate the DDS nodes on one or more networks. A point-to-point network functions in a similar way to most point-to-point message-oriented systems. In this deployment scheme the DDS offers some QoS features. If the implementer requires true real-time QoS features, then to fully realize the power of DDS - a UDP based network is required. In effect there are three schemes that can be operated over one or more network segments; these options are listed in the least to the most scalable in terms of real-time capabilities:

- TCP/IP point-to-point between peers
- UDP point-to-point between peers
- UDP multicast between peers

The mechanism to integrate OMG with SOS is to use the SOS or SensorML/Observations and Measurements schema to generate IDL (native DDS format) using tools. Alternatively we can embed the SOS XML into the payload directly or using a compressed encoding scheme such as FastInfoSet, ASN.1 PER, or a similar XML encoding technology. Note that some of the metadata must remain visible as strings to enable a feature called content-based routing and certain types of query-based topics. To the extent that the SOS data is visible to the IDL data structure DDS supports these advanced features.

The provision of QoS on a per-entity basis is a significant capability provided by DDS. Being able to specify different QoS parameters for each individual Topic, Reader or Writer gives developers a large palette from which to design their system. This is the essence of data centricity within DDS.

4.6 Recommendations and Conclusions

Implementers of distributed and embedded systems that interface with SOS should try to ensure they provide sufficient SOS data and meta-data artifacts. In most cases this integration must be performed manually. In some OMG DDS implementations, the SOS data models such as the OGC Observations and Measurements and OGC SensorML data models can be automatically mapped to efficient and routable IDL data structures. This approach ensures both compliance and scalability.

Implementers who are building a new system have several choices. The NSF DataTurbine software can be used to build a high-performance distributed system with asynchronous push capabilities. The implementer should consider the role of SOS HTTP GET (REST) integration and the role of specialized channels for real-time SOS data using the native protocol.

The OMG DDScan be used to build high-performance distributed system or even an embedded system. DDS is used in a wide range of applications from embedded ship-board and avionics, robotics, industrial control, radar tracking, etc. DDS Tools can be

used to translate the OGC XML-Schema based content model into Interface Definition Language (IDL) to provide a more efficient and routable encoding mechanism.

Distributed Systems Links:

1. OGC SOS: http://www.opengeospatial.org/standards/sos

2. IANA MIME Media Types: http://www.iana.org/assignments/media-types/

3. Unidata IDD: http://www.unidata.ucar.edu/software/idd/

4. OOI CI: http://oceanobservatories.org

5. NSF DataTurbine: http://www.dataturbine.org/

6. OMG DDS: http://www.omg.org/spec/DDS/

7. JINI: http://www.jini.org/wiki/Main_Page

8. Roy Fielding, 2008-10-20, http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

9. Tony Cook, UAH, Unidata IDD/LDM SOS Integration, http://sos-ws.tamu.edu/tethys/tabs?request=GetCapabilities&service=SOS&version=1.0.0

5. Topic: CSW Registry

5.1 Goal

This team had the following goal of implementing a standardized OGC Catalog Service for Web (CSW) to provide registration for SOS services, with the following objectives:

- Creation of ISO19115 / ISO19139 metadata profile for SOS implementations.
- Automatic harvesting of SOS service metadata via GetCapabilities.
- A web interface for registration by only submitting the SOS get Capabilities.
- Incorporation of testing functionality while registering and SOS.

5.2 Motivation

The marine community has not yet standardized on a standard registry component. OOSTethys defined a registry component and implemented one in Phase I, but it was not based on standards. Participants at the GEOSS Implementation pilot project were implementing a CSW. In Phase II OOSTethys members wanted to advance a CSW implementation that will facilitate to register SOS, since there was not one available, and other projects were advancing standards that can served as a reference.

5.3 Participants

- UAH - Manil Maskey
- SURA - Luis Bermudez
- GOMOOS - Eric Bridger

5.4 Discussion

This section describes the work advance at the Oceans IE Phase II related to CSW service registry. This particular implementation of the CSW standard shields service providers from being burdened with learning complex XML specifications; hence, making the registration process very easy for them in addition to provide a robust validation process. The two components that we have implemented are XSLT transformation from SOS metadata to ISO 19119 and incorporation of testing via TEAM Engine integration.

5.4.1 SOS Metadata to ISO 19119

The team decided to use Deegree (http://www.deegree.org/) as the implementation for implementation of the catalog service for web (CSW). The CSW transaction method only accepts ISO 19119 service metadata record for registration. We use XSLT transformation to automatically harvest of all necessary information from the SOS GetCapabilities response into the registry to ISO 19119,

5.4.2 TEAM Engine integration

OGC has developed a test suite TEAM Engine for validation of OGC service specifications. It is available as an opens ource sfotware at the OGC web site (http://cite.opengeospatial.org/). TEAM Engine was slightly modified and integrated integrate into the registry. This integration allows for a robust validation of the SOS during registration. The web interface notifies the service providers whether their service is compliant with OGC SOS specifications. The CSW Registry – 1 Figure sumamrised the process.

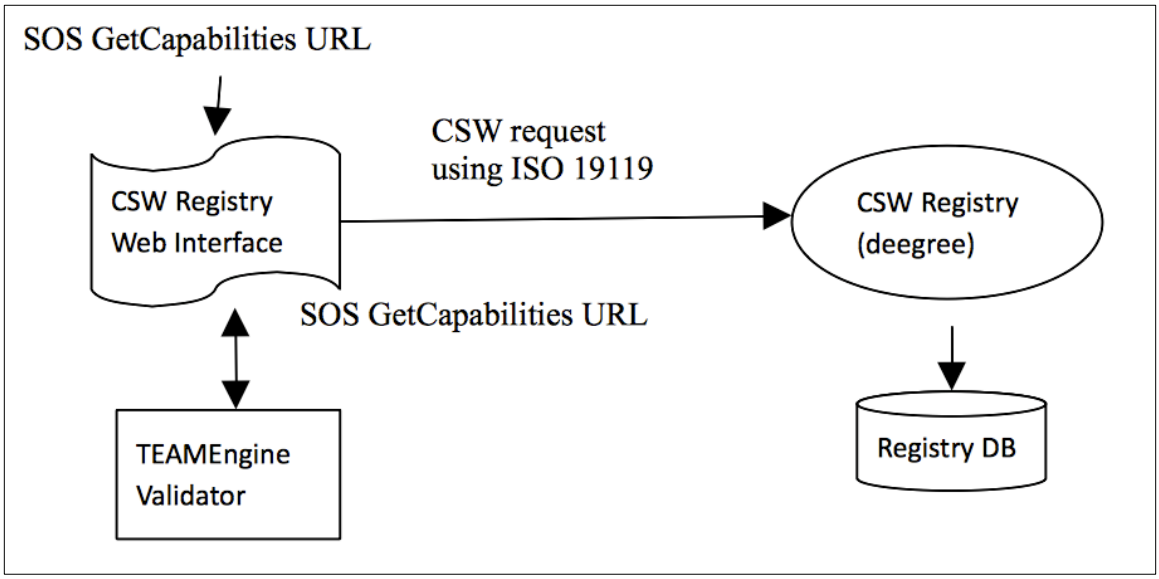


Figure [CSW Registry - 1]. This figure illustrates the current mechanism for registering an SOS in the OOSTethys/OceansIE registry.

5.5 Recommendations and Conclusions

- Services should be tested and validated at the moment of registration.
- The metadata required to publish a CSW can be extracted from the SOS getCapabilities document; furthermore, if metadata is missing to created the CSW it should be added to the getCapabilities metadata.

6. Topic: Semantic Registry and Services

6.1 Goals

This team had the following goals:

- Determine and implement features at the MMI Ontology Registry and Repository that exploit the semantic information associated with data registry and observation services.

- Determine required vocabularies/ontologies to support the association of references to corresponding definitions in SWE documents.

6.2 Motivation

Semantic mediation is a required mechanism to allow system interoperability and data integration. Such mechanism comprises a set of key operations including controlled vocabulary definition and maintenance, terminology mappings, and inference, among others. The Marine Metadata Interoperability Project has advanced the MMI Ontology Registry and Repository (MMI ORR), a system that provides semantic services for the marine and earth science communities.

In Oceans IE Phase I, semantic mediation was demonstrated by providing a searching capability via general categories for phenomena (See Figure: Semantic Registry and Services-1). Using semantic web technology services were categorize appropriately.

A complementary MMI service provides URI resolution for both vocabularies and individual entities defined within. This team focused on enhancing services at the MMI ORR to enable semantic information in data registry and observation services. Two main components–URN support and web resolution; and semantically-enabled generation of sensor system formal descriptions–are described below.

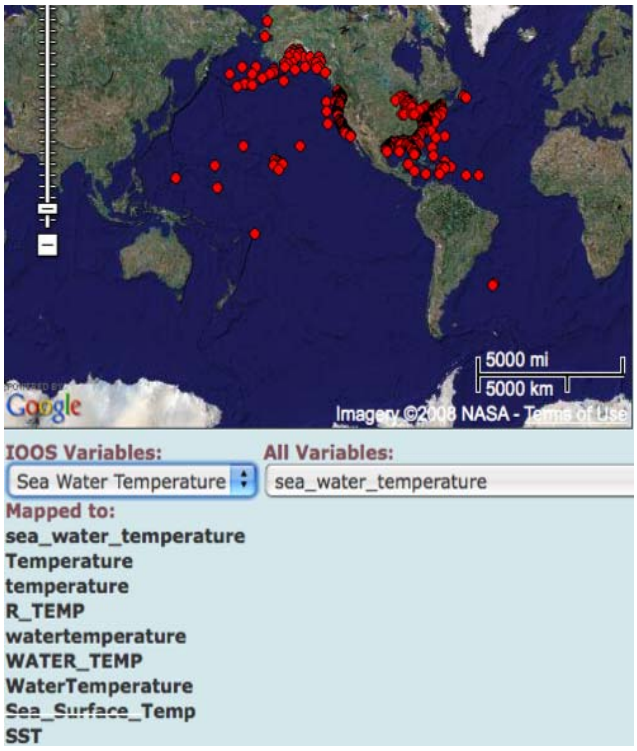


Figure [Semantic Registry and Services-1]. Categorization of SOS services. This example demonstrates the categorization of services which were tagged with terms narrower or same-as Sea Water Temperature.

6.3 Participants

- MBARI/MMI: Carlos Rueda
- MMI: John Graybeal
- SURA: Luis Bermudez
- GoMOOS: Eric Bridger
- NOAA: Jeff DLB

- UAH: Tony Cook

6.4 Discussion

6.4.1 Support for Uniform Resource Names and web resolution

Uniform Resource Identifiers (URIs) are one of the key enabling technologies for the semantic web, which is a core technology used by the MMI ORR. Vocabulary terms and their relationships are identified with URIs so they can be used and interlinked globally. There are two URI subcategories: Uniform Resource Locators (URL), which, besides being identifiers, are in principle also intended to provide a mechanism to access the identified resource; and Uniform Resource Names (URN), whose main purpose is to identify but not necessarily to locate resources. The MMI ORR was providing only support for URLs as the preferred identification mechanism (since an important feature is that the generated identifiers be also immediately resolvable). However, there was a need to also support URNs, which are used by some communities, (e.g. OGC). Since URNs are not self-resolvable, a mechanism for their resolution was needed. The implemented mechanism is described below.

The MMI ORR allows to create an ontology from a table of definitions, for example:

name	description
foo	foo description ...
baz	baz description ...

The table can be populated by importing text in CSV format. The first row specifies properties for the terms in the subsequent rows. First column is special in that it is used to create the URI for the term in each row. Assuming the URI of the vocabulary as a whole is http://mmisw.org/ont/myvocab, then the final URI for the "foo" term will be "http://mmisw.org/ont/myvocab/foo". In this example, once the vocabulary is registered at the ORR, all these URIs will be directly resolvable.

For the Oceans IE Phase II, the following mechanism was implemented to allow the user to completely specify the final URI (URL or URN) for each term. If the header label of the first column in the term table is "URI", then the values in the column will be used exactly as given. In this case no automatic creation of URIs will be performed by the MMI OOR. For example:

URI	description
http://mydomain.xyz/abcd/foo	foo description ...
http://other.xyz/baz	baz description ...

In particular, the given URIs can be URNs, for example:

URI	description
urn:ogc:def:crs:ogc:1_3:crs27	NAD27 longitude-latitude B.5 in OGC 06-04

As noted above, any ORR-generated URLs for terms and vocabularies are directly resolvable. Another provided mechanism to resolve any registered URI, which is particularly useful for URNs, is via an HTTP request with the “uri” parameter to the http://mmisw.org/ont service, for example:

```
http://mmisw.org/ont?uri=urn:ogc:def:crs:ogc:1_3:crs27
```

By default, the format of the response will be determined according to content negotiation, for example, in RDF/XML if the client indicates "application/rdf+xml" as the preferred format. A 'form' parameter can also be used to explicitly request a particular format, for example:

```
http://mmisw.org/ont?uri=urn:ogc:def:crs:ogc:1_3:crs27&form=rdf
http://mmisw.org/ont?uri=urn:ogc:def:crs:ogc:1_3:crs27&form=html
```

6.4.2 Semantically-enabled generation of sensor system descriptions

Sensor Observation Services (SOS), developed by the OGC Sensor Web Enablement (SWE) initiative, provides an interface for discovering, binding to, and interrogating individual sensors, instruments, platforms, and systems. Soft-typing characteristics of the associated SWE model languages offer a means to augment the descriptions with rich semantics. Figure [Semantic Registry and Services-2] is a SensorML document fragment showing an observation offering with embedded URIs. Each URI corresponds to a concept defined in an ontology and available through MMI ORR services. When resolved, these URIs provide rich semantics to the corresponding elements in the offering.

```
– <sos:ObservationOffering gml:id="observationOffering_1455">
    <gml:description/>
  – <gml:boundedBy>
    – <gml:Envelope srsName="urn:ogc:def:crs:EPSG:6.5:4326">
        <gml:lowerCorner>36.69623 -122.39965</gml:lowerCorner>
        <gml:upperCorner>36.69623 -122.39965</gml:upperCorner>
      </gml:Envelope>
    </gml:boundedBy>
  – <sos:time>
    – <gml:TimePeriod gml:id="timePeriod3">
        <gml:beginPosition>2008-06-09T09:36:19Z</gml:beginPosition>
        <gml:endPosition>2008-06-10T02:06:21Z</gml:endPosition>
      </gml:TimePeriod>
    </sos:time>
    <sos:procedure xlink:href="urn:mbari:org:device:1455"/>
    <sos:observedProperty xlink:href="http://mmisw.org/ont/cf/parameter/sea_water_temperature"/>
    <sos:observedProperty xlink:href="http://mmisw.org/ont/cf/parameter/conductivity"/>
    <sos:observedProperty xlink:href="http://mmisw.org/ont/cf/parameter/pressure"/>
    <sos:observedProperty xlink:href="http://mmisw.org/ont/cf/parameter/sea_water_salinity"/>
    <sos:featureOfInterest xlink:href="http://mmisw.org/mmi/20080516/system#EarthRealm"/>
    <sos:responseFormat>text/xml; subtype="om/1.0.0"</sos:responseFormat>
    <sos:responseMode>inline</sos:responseMode>
  </sos:ObservationOffering>
```

Figure [Semantic Registry and Services-2]. Example of a SOS observation offering with embedded URIs for semantic augmentation of the described service.

Once these descriptions are in place, both syntactic and semantic interoperability are facilitated. However, integrating the necessary elements during the creation of these descriptions is a challenging task for many data managers and users. The goal of this component was to implement a web tool to facilitate the creation of these formal documents, in particular SensorML, with seamsless integration of semantic definitions from the MMI ORR.

APIs were developed on the MMI ORR to support the development of semantically-enabled tools. With these interfaces, we developed a simple SensorML web Generator, available at http://mmisw.org/smlmor. The user indicates the structure of the sensor system (system type, variables, and subsystems) while being able to choose URIs via drop-down lists containing standard entries for sensor types and variables. The drop-down lists are populated with definitions registered in the MMI ORR. Figure [Semantic Registry and Services-3] illustrates the basic interaction with the definition of an output variable. The user clicks a button to select an appropriate definition from the NetCDF Climate and Forecast (CF) Metadata Convention standard name vocabulary (http://cf-pcmdi.llnl.gov/). A similar selection mechanism is available for sensor types. The tool allows the description to include nested subsytems, each with the corresponding variables. Once the desired structure has been completed, the "Generate SensorML" button creates the resulting SensorML definition.

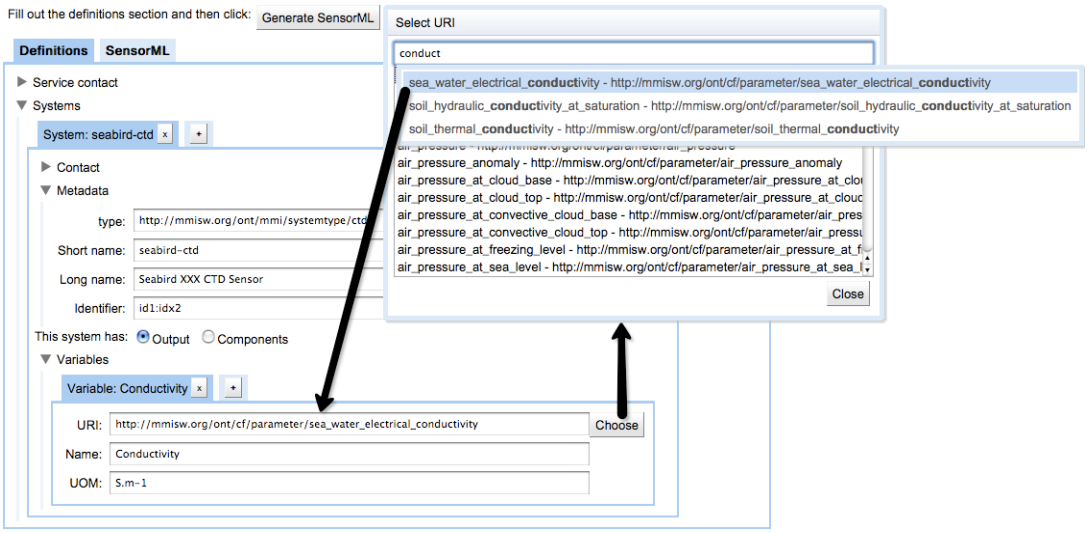


Figure [Semantic Registry and Services-3]. Interaction mechanism with the MMI ORR in the SensorML Generator interface.

6.5 Recommendations and Conclusions

The MMI Ontology Registry and Repository (Rueda et al, 2009), provides a key support for semantic interoperability, including term mappings, semantic queries and inferencing. Two functionalities were advance in Oceans IE Phase II: URI resolution for non URLs and a web tool to create standard documents with semantic annotations available at MMI OOR. This way OGC Sensor Web Enablement services can be enriched with semantic

references that are resolvable against the MMI ORR. The vocabularies used during the experiment (although some of them still in preliminary form) demonstrated the benefits of easily linking semantic information to sensor descriptions.

Concrete recommendations are as follows:

- Data and sensor managers should continue evaluating existing vocabularies and determining needs toward agreed upon definitions according to their metadata management requirements. For example, sensor and platform types.
- Workshops and other forms of training and outreach should be offered to data users and managers to promote the use of semantic web technologies, and demonstrate its benefits especially in terms of interoperability to the community at large.
- Ontology registries like the MMI ORR and similar vocabulary servers should be leveraged, supported, and advertised to gain broader exposure and thus get valuable feedback for continued improvement.

Rather than technical, a main challenge remains regarding the engagement of various communities around common semantic approaches and unified strategies. Key aspects include semantic registries, federated vocabulary repositories, and common APIs across diverse ontology and vocabulary servers. We finally recommend that immediate efforts continue to address these challenges, ideally in as a comprehensive way as possible.

More information about the MMI Ontology Registry and Repository and its associated semantic services can be found at http://marinemetadata.org/mmiorrusrman/.

References

Rueda, C., Bermudez, L., Fredericks, J. The MMI Ontology Registry and Repository: A Portal for Marine Metadata Interoperability. MTS/IEEE Oceans'09. Biloxi, Mississippi. October, 2009.

7. Topic: Complex Systems

7.1 Goals

This team had the following goal:

Advance SOS encoding for systems that contain other systems. In particular, collection of stations (could be heterogeneous), and platforms containing multiple sensors. The encoding/ serving mechanism should allow to perform time-spatial queries over these complex system and provide the relation of the system to its components.

7.2 Motivation

An ocean observing system could be defined as a set of independent elements that interact to form a whole for the purpose of observing ocean data. SensorML defines that a sensor is a system. Therefore it is possible for and SOS to provide system observations. Questions that can arise when trying to implement and SOS are: Is a buoy a system? Is a collection of stations a system? This topic clarifies the concept of “systems” and provides recommendations about how to encode complex systems.

7.3 Participants

- SURA - Luis Bermudez
- dBscale Sensing Technologies - Eric Delory
- MBARI - Tom O'Reilly
- Technical University of Catalonia - Joaquin del Rio Fernandez

7.4 Discussion

This topic discuses several observing system components and their relation. The definition and the relation to the SensorML system conceptual model are provided in this section. The relationship between different components are depicted in Figure [Complex Systems-1], and the remainder of the section provides the definition for each component in the figure.

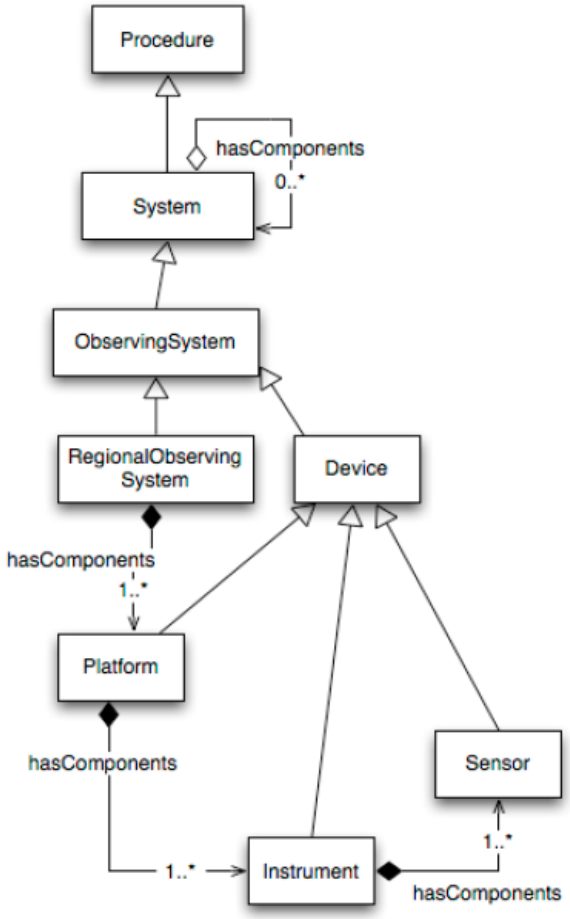


Figure [Complex Systems-1]. Simple UML for systems components. The top classes depict the SensorML conceptual model where a system is a procedure. And a system has components.

7.4.1 Sensor

Based on the IEEE 1451 definition, a **sensor** is a device that converts a physical, chemical, or biological parameter into an electrical signal, which is ultimately output to an observing system as data. Common examples include temperature, conductivity, or solar radiation sensors.

A sensor is the most basic unit. It does not contain other systems. The description of a sensor requires one system description within a SensorML document. The description of a sensor in IEEE 1451 take place in the Transducer Electronic Data Sheet (TEDS). Metadata information about the sensor could be transmitted to SOS enabled services by extracting the information from an STWS. SensorML or TEDS definitions could be updated on the instrument using PUCK.

The observation offering of a sensor is composed of the output of the sensor channel. It represents the value of one parameter. The location and time of the observation is inherit from the instrument / platform where this sensor is located.

7.4.2 Instrument

An **instrument** is a physical collection of multiple sensors that a manufacturer has integrated into a single device. For example a CTD aggregates conductivity, temperature, and depth sensors. The instrument has an external physical interface through which the instrument and its sensors can be operated and the sensor data retrieved.

The instrument metadata requires that various components are specified within SensorML. All the components could be describe in one SensorML or several SensorML documents. However the description of the instrument must specify the containment of the other components, either pointing to external SensorML or to other parts of the same SensorML document.

An instrument could advertise one observation offering per sensor and additionally one observation offering that aggregates all the sensor offerings. Similarly to the sensor, some instruments have no location and time sensors. For such sensors to advertise an instrument offering, it is necessary to get the location and time data from another instrument in the platform within the same context (e.g., a GPS).

7.4.3 Platform

A **platform** physically aggregates multiple instruments and interacts with them through their interfaces. The platform is usually integrated by the organization that deploys it, and can also provide infrastructure for instrument power, data storage and telemetry, and other functions. Platforms can be stationary (e.g. a seafloor observatory node), may drift (e.g. a mooring) or may be actively mobile (e.g. an autonomous underwater vehicle). The description of the platform in SWE uses the same pattern as previously described for an instrument.

When advertising observations for a platform one observation could exist aggregating all the output of all the instruments that are important to the end user. For example, an

offering could have time-interpolated values for different instruments. Also, other offerings could exist, one for each instrument and also one for each sensor.

7.4.4 Regional Observing System

Finally, a **regional observing system** is comprised of multiple platforms in a defined region, and often defines common protocols and procedures to operate the constituent platforms and instruments as well as process the instrument data. That is, instruments within an observing system may be interoperable with one another.

This is the most complex system. Composition could be handled the same way as described for instruments and platforms. In this case the location of the system is usually described as the bounding box that covers the entire area of all the platforms it contains. Similar patterns as described in the platform use case apply to regional observing systems when using SensorML and configuring observation offerings.

There could be different types of regional Observing Systems. For example there could be a coastal observatories (e.g. MBARI, LEO-15 or MBARI's MOOS), a Regional Association (e.g.,CenCOOS), or the U.S. (IOOS).

7.5 Recommendations and Conclusions

The results of this group were published and presented at the Oceans 2009 conference (Bermudez, et. al., 2009). The concrete recommendations are as follows:

- In the getCapabilities create an offering per instrument (e.g. CTD. MetSYS, etc..)
- In the getCapabilities create an offering that logically aggregates the observations (e.g Mooring). It should contain the extended BBOX and should describe all the outputs.
- Create a SensorML per instrument (e.g. CTD).
- Create a SensorML per platform, and describe the grouping (Mooring has CTD, Mooring has MetSYS, etc..).
- If proving a regional system, create a SensorML per region, grouping the platforms, within the region.
- The getObservation respond for a complex system (e.g. platform) will be an observation collection, where each observation member is an observation per instrument.
- To enable a query containing latitude longitude and time for all the observations of an observatory just put in one offering all the observatory observations as a complex system.

8. Topic: Large number of Observation Offerings

8.1 Goals

Create an implementation of a complex system with large number of offerings.

8.2 Motivation

Querying an entire region for observations of interest can be complicated if the region contains large number of sensors. Even though a system can be offered as a one complex

system, as discussed in section 9, it is also useful to provide individual offerings per sensor. We wanted to test performance issues encountered when dealing with large numbers of offerings.

8.3 Participants

- SURA - Luis Bermudez
- UAH - Tony Cook

8.4 Discussion

METAR is a standard format used worldwide for reporting meteorlogical data from weather stations. The station data may be augmented by trained human observers as well. Reports are typically issued hourly, but may be updated more frequently if conditions change rapidly, or unusual weather occurs. The number of potential reporting stations is quite large (8,394 at the time of the initial SOS deployment). A list of stations is maintained at the following URL:

http://www.rap.ucar.edu/weather/surface/stations.txt

Texas A&M receives raw METAR observations via a Unidata IDD feed. The METAR records are parsed and recorded into a PostgreSql Database. When designing the SOS for serving this data, one of the first concerns was how to structure the ObservationOfferings. A brief synopsis of possibilities and their advantages and disadvantages follow.

- **One ObservationOffering per station**: One SOS- In this approach, every station is an ObservationOffering in a single SOS. Stations can be queried individually, and spatial queries can be performed for all stations in a requested geographic bounding box. Individual ObservedProperties can also be queried for any station. However, for a large number of stations, the size of the SOS Capabilities document can grow prohibitively large. A simple Capabilities file created using all 8,394 stations in the above URL was over 12 Megabytes.
- **Geographically-grouped stations**: One SOS – In this approach, sets of geographically proximal stations can be grouped into individual ObservationOfferings. For instance, there could exist 50 ObservationOfferings in an SOS for US Metar data, where each offering contains all the stations in a state. The resulting Capabilities document is on the order of a few hundred kilobytes, which is much more manageable from both client and server perspective. The drawback to this approach that individual stations cannot be queried.

8.5 Recommendations and Conclusions

The recommendation from this group is to provide one Observation Offering per station and one observation offering grouping multiple geographically-related stations.

This is a hybrid approach between the two discussed in the previous section. Here, different SOS endpoints would exist for different geographic regions, with a manageable number of stations grouped into each instance. Each station is then packaged as an individual ObservationOffering. For example, there could be one SOS for serving Southeast U.S. Stations, which would contain a few hundred stations. Any individual

station is then queryable. The maintenance of multiple SOS instances is a small complicating factor for the provider, but should be manageable.

9. Topic IEEE 1451- OGC SWE Harmonization

9.1 Goals

Within the IEEE1451-OGC/SWE integration workplan, the task reported in this chapter is to evaluate and refine integration of IEEE 1451-TEDS and OGC SML. TEDS and SML are respectively the sensor/instrument metadata containers of both suites of standards. Outcome will include recommendations to revise IEEE145-TEDS XML and OGC-SML specifications, or both.

9.2 Motivation

A IEEE1451 – SWE integration testbed was collaboratively developed by ESONET, NIST, OGC, MBARI, Northrop Grumman, Compusult, GoMOOS. Development of that testbed identified several areas that should be addressed, including Item 1. Integration of IEEE-1451 TEDS and SensorML, and Item 2. Asynchronous event notification between IEEE-1451 and OGC-SWE. The work implies collaboration of developers and the respective standard editing teams. In the OIE phase 2 was initiated item 1., with the motivation to progressively respond to other harmonizing needs in future experiments.

9.3 Participants

- ESONET: Eric Delory
- dBscale Sensing Technologies: Jose Mendoza
- UPC: Joaquin Del Rio (
- University of Münster: Simon Jirka
- U. Bremen/MARUM: Christoph Waldmann
- NIST: Kang Lee, Yuyin Song
- MBARI: Tom O'Reilly
- SURA: Luis Bermudez

9.4 Discussion

The harmonization was performed using X queries that created a SensorML from IEE 1451 documents. Details are available in 9.4.2. Section 9.4.1 presents a discussion of one of the Units of Measure harmonization, to exemplify the type of output that is looked forward in this task

9.4.1 TEDS-SML harmonizing: Example of Units of Measure

SensorML

Quantity and count are the two objects to include a uom property for specifying units of measure. One can specify units of measure in one of three ways. The first is to use the xlink:href attribute to reference known unit definitions that have been defined online using gml:UnitDefinition, as in the following example for parts-per-million.

```
<swe:uom xlink:href="urn:ogc:def:unit:OGC:ppm"/>
```

The second is to utilize the gml:UnitDefinition object to define the units inline. This would typically be used when one wishes to define a complex unit of measure that is perhaps not available in a standard unit dictionary, as in the following example for slugs per foot-second.

```
<gml:ConventionalUnit gml:id="slug_fts">
    <gml:name>slugs/foot-second</gml:name>
    <gml:name codeSpace="urn:ogc:tc:arch:doc-rp(05-010)">
urn:ogc:def:uom:OGC:slug_fts</gml:name>
    <gml:quantityType>dynamic viscosity</gml:quantityType>
    <gml:catalogSymbol>slug/fts</gml:catalogSymbol>
    <gml:conversionToPreferredUnit uom="#Pa.s">
    <gml:factor>47.9</gml:factor>
    </gml:conversionToPreferredUnit>
    <gml:derivationUnitTerm uom=" urn:ogc:def:unit:OGC:slug" exponent="1"/>
    <gml:derivationUnitTerm uom=" urn:ogc:def:unit:OGC:ft" exponent="-1"/>
    <gml:derivationUnitTerm uom=" urn:ogc:def:unit:OGC:s" exponent="-1"/>
</gml:ConventionalUnit>
```

The third option for specifying units of measure is to utilize the Unified Code for Units of Measure (UCUM) within the code attribute of uom, as in the following example for centimeter-squared per second.

```
<swe:uom code="cm2.s-1"/>
```

IEEE1451

In IEEE1451, there seems to be no currently publically available examples of TEDS that include a clear description of the syntax to encode units of measure in XML TEDS (schema is available for base units here: http://grouper.ieee.org/groups/1451/0/1451HTTPAPI/SmartTransducerDataModel.xsd)

Though Unit expressions leave no room for ambiguity in the 1451.0 standard as they shall in all possible cases be based on SI base and derived units (ISO 1000), that is, when possible, as some units may not be derivable from that set.

"Derived units are expressed algebraically in terms of base units. […] The symbols for derived units are obtained by means of the mathematical operations of multiplication and division. For example, the derived unit for molar mass (mass divided by amount of substance) is the kilogram per mole that has the symbol kg/mol."

Although units can also be extended by a Units Extension Data Block, this is to provide more detailed information on what is being measured, so this is irrelevant here as this block does not provide a description or definition of the unit itself.

It appears that machine-readability of SML and TEDS instances suffer from the diversity of structure (parsing issues), and naming possibilities, but in both standards there is in most cases a way to encode units from SI base units (through mandatory derivations in IEEE1451 and UCUM in SWE Common). This is the most adequate approach so long as both types of standard instances provide a description of derivations. As UCUM (used in SWE common and SML through the code attribute) is also used in ISO 19136 and ISO 19136 includes SI base unit derivation we hereby recommend to use the ISO19136 schema for units available here:

http://standards.iso.org/ittf/PubliclyAvailableStandards/ISO_19136_Schemas//units.xsd

or up to now, a.k.a

http://schemas.opengis.net/gml/3.2.1/units.xsd

According to S@NY D.2.2.1 's Sensor Taxonomy: *"The Unified Code for Units of Measure (UCUM) from Gunther Schadow at the Regenstrief Institute for Health Care and Indiana University School of Medicine proposes both case-sensitive and case-insensitive symbols for a large suite of units of measure. In general these follow the precedents set by earlier standards (e.g. ISO 31, NIST Special Publication 811, ISO 2955, ANSI X3.50), with a small number of adjustments to remove ambiguities. All base and derived SI units have their usual symbol (e.g. meter = m, second = s, watt = W plus prefix: mW, uA, ns, mrad)."*

9.4.2 TEDS-SML Mapping

Once harmonizing issues are resolved, a mapping can be performed. Following illustrates the mapping of TEDS XML to SensorML for a CTD, using xquery. This xquery file can be easily embedded in a java code for example.

Xquery example of TEDS to SML instance mapping for a CTD Instrument:

```
declare namespace a = "http://www.opengis.net/sensorML/1.0.1";
declare namespace b = "http://www.opengis.net/gml";
declare namespace c = "http://www.opengis.net/swe/1.0.1";

<a:SensorML>
  <a:member>
    <a:System>
      <b:description>
        {
        doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/ProductDescription/text()
        }
      </b:description>
      <a:keywords>
        <a:KeywordList>
          {
          for $Keyword1 in doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Keywords/Keyword
          return
          <a:keyword/>
          }
        </a:KeywordList>
      </a:keywords>
      <a:identification>
        <a:IdentifierList>
          <a:identifier name="UID">
            <a:Term definition="urn:ogc:def:identifier:OGC:uuid">
              <a:value>
                {
                doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/timId/text()
```

```
            }
         </a:value>
      </a:Term>
   </a:identifier>
   <a:identifier name="Short Name">
      <a:Term definition="urn:ogc:def:identifier:OGC:shortName">
         <a:value>
            {
            doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/GroupName/text()
            }
         </a:value>
      </a:Term>
   </a:identifier>
   <a:identifier name="Long Name">
      <a:Term definition="urn:ogc:def:identifier:OGC:longName"/>
   </a:identifier>
   <a:identifier name="Manufacturer Name">
      <a:Term definition="urn:ogc:def:identifier:OGC:manufacturerName">
         <a:value>
            {
            doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/ManufacterId/text()
            }
         </a:value>
      </a:Term>
   </a:identifier>
   <a:identifier name="Model Number">
      <a:Term definition="urn:ogc:def:identifier:OGC:modelNumber">
         <a:value>
            {
            doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/ModelNo/text()
            }
         </a:value>
```

```
            </a:Term>
         </a:identifier>
         <a:identifier name="Serial Number">
            <a:Term definition="urn:ogc:def:identifier:OGC:serialNumber">
               <a:value>
                  {
                  doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/SerialNo/text()
                  }
               </a:value>
            </a:Term>
         </a:identifier>
         <a:identifier name="Device ID">
            <a:Term definition="urn:ogc:def:identifier:ESONET:deviceID"/>
         </a:identifier>
      </a:IdentifierList>
   </a:identification>
   <a:validTime>
      <b:TimePeriod>
         <b:beginPosition>
            {
            doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/PeriodTime/BeginDate/text()
            }
         </b:beginPosition>
         <b:endPosition>
            {
            doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/PeriodTime/EndDate/text()
            }
         </b:endPosition>
      </b:TimePeriod>
   </a:validTime>
   <a:capabilities name=""/>
   <a:contact arcrole="urn:ogc:def:classifiers:OGC:contactType:manufacturer">
```

```
<a:ResponsibleParty>
   <a:organizationName>
      {
      doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/contacts/manufacturer/OrganizationName/text()
      }
   </a:organizationName>
   <a:contactInfo>
      <a:phone>
         <a:voice>
            {
            doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/contacts/manufacturer/PhoneVoice/text()
            }
         </a:voice>
         <a:facsimile>
            {
            doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/contacts/manufacturer/Facsimile/text()
            }
         </a:facsimile>
      </a:phone>
      <a:address>
         <a:deliveryPoint>
            {
            doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/contacts/manufacturer/DeliveryPoint/text()
            }
         </a:deliveryPoint>
         <a:city>
            {
            doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/contacts/manufacturer/City/text()
            }
         </a:city>
         <a:administrativeArea>
            {
```

```
                        doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/contacts/manufacturer/AdministrativeArea/text()
                        }
                    </a:administrativeArea>
                    <a:postalCode>
                        {
                        doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/contacts/manufacturer/PostalCode/text()
                        }
                    </a:postalCode>
                    <a:country>
                        {
                        doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/contacts/manufacturer/Country/text()
                        }
                    </a:country>
                    <a:electronicMailAddress>
                        {
                        doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/contacts/manufacturer/email/text()
                        }
                    </a:electronicMailAddress>
                </a:address>
                <a:onlineResource
href="{doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/contacts/manufacturer/OnlineResource}"/>
            </a:contactInfo>
        </a:ResponsibleParty>
    </a:contact>
    <a:contact arcrole="urn:ogc:def:classifiers:OGC:contactType:owner">
        <a:ResponsibleParty>
            <a:organizationName>
                {
                doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/contacts/owner/OrganizationName/text()
                }
            </a:organizationName>
            <a:contactInfo>
```

```
<a:phone>
   <a:voice>
      {
      doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/contacts/owner/PhoneVoice/text()
      }
   </a:voice>
   <a:facsimile>
      {
      doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/contacts/owner/Facsimile/text()
      }
   </a:facsimile>
</a:phone>
<a:address>
   <a:deliveryPoint>
      {
      doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/contacts/owner/DeliveryPoint/text()
      }
   </a:deliveryPoint>
   <a:city>
      {
      doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/contacts/owner/City/text()
      }
   </a:city>
   <a:administrativeArea>
      {
      doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/contacts/owner/AdministrativeArea/text()
      }
   </a:administrativeArea>
   <a:postalCode>
      {
      doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/contacts/owner/PostalCode/text()
      }
```

```
            </a:postalCode>
            <a:country>
               {
               doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/contacts/owner/Country/text()
               }
            </a:country>
            <a:electronicMailAddress>
               {
               doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/contacts/owner/email/text()
               }
            </a:electronicMailAddress>
         </a:address>
         <a:onlineResource href="{doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/contacts/owner/OnlineResource}"/>
      </a:contactInfo>
   </a:ResponsibleParty>
</a:contact>
<a:documentation>
   <a:Document>
      <b:description>
         {
         doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Documentation/Description/text()
         }
      </b:description>
      <a:date>
         {
         doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Documentation/Date/text()
         }
      </a:date>
      <a:format>
         {
         doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Documentation/Format/text()
         }
```

```
      </a:format>
      <a:onlineResource>
        {
        doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Documentation/UrlDoc/text()
        }
      </a:onlineResource>
    </a:Document>
</a:documentation>
<a:position name="stationPosition">
    <b:Vector>
      <b:coordinate name="Latitude">
        <b:Quantity axisID="Y">
          <b:uom code="{doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Position/Latitude/UOM}"/>
          <b:value>
            {
            doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Position/Latitude/Value/text()
            }
          </b:value>
        </b:Quantity>
      </b:coordinate>
      <b:coordinate name="longitude">
        <b:Quantity axisID="X">
          <b:uom code="{doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Position/Longitud/UOM}"/>
          <b:value>
            {
            doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Position/Longitud/Value/text()
            }
          </b:value>
        </b:Quantity>
      </b:coordinate>
      <b:coordinate name="altitude">
        <b:Quantity>
```

```
            <b:uom code="{doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Position/Altitude/UOM}"/>
            <b:value>
               {
               doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Position/Altitude/Value/text()
               }
            </b:value>
         </b:Quantity>
      </b:coordinate>
   </b:Vector>
</a:position>
<a:interfaces>
   <a:InterfaceList>
      {
      for $Interface in doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Interfaces/Interface
      return
      <a:interface name="{$Interface/Name}">
         <a:InterfaceDefinition>
            <a:physicalLayer>
               <c:Category>
                  <b:description>
                     {
                     $Interface/Description/text()
                     }
                  </b:description>
               </c:Category>
            </a:physicalLayer>
         </a:InterfaceDefinition>
      </a:interface>
      }
   </a:InterfaceList>
</a:interfaces>
<a:inputs>
```

```
            <a:InputList>
               <a:input>
                  <b:ObservableProperty
definition="{doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Inputs/Temperature/Definition}">
                     <c:description>
                        {
                        doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Inputs/Temperature/Description/text()
                        }
                     </c:description>
                     <c:name>Temperature</c:name>
                  </b:ObservableProperty>
                  <b:ObservableProperty>
                     <c:name>Conductivity</c:name>
                  </b:ObservableProperty>
               </a:input>
               <a:input>
                  <b:ObservableProperty
definition="{doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Inputs/Conductivity/Definition}">
                     <c:description>
                        {
                        doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Inputs/Conductivity/Description/text()
                        }
                     </c:description>
                  </b:ObservableProperty>
               </a:input>
               <a:input>
                  <b:ObservableProperty
definition="{doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Inputs/Preassure/Definition}">
                     <c:description>
                        {
                        doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Inputs/Preassure/Description/text()
                        }
```

```
            </c:description>
          </b:ObservableProperty>
        </a:input>
      </a:InputList>
    </a:inputs>
    <a:outputs>
      <a:OutputList>
        <a:output>
          <b:Quantity definition="urn:ogc:def:property:OGC:temperature">
            <c:name>temperature</c:name>
            <b:uom>
              {
              doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Outputs/Temperature/uom/text()
              }
            </b:uom>
            <b:constraint>
              <b:AllowedValues>
                <b:interval>
                  {
                  doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Outputs/Temperature/interval/text()
                  }
                </b:interval>
              </b:AllowedValues>
            </b:constraint>
            <b:value>
              {
              doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Outputs/Temperature/value/text()
              }
            </b:value>
          </b:Quantity>
          <b:Quantity definition="urn:ogc:def:property:OGC:conductivity">
            <c:name>conductivity</c:name>
```

```
    <b:uom>
      {
      doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Outputs/Conductivity/uom/text()
      }
    </b:uom>
    <b:constraint>
      <b:AllowedValues>
        <b:interval>
          {
          doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Outputs/Conductivity/interval/text()
          }
        </b:interval>
      </b:AllowedValues>
    </b:constraint>
    <b:quality>
      <b:Text>
        <b:value>
          {
          doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Outputs/Conductivity/value/text()
          }
        </b:value>
      </b:Text>
    </b:quality>
  </b:Quantity>
  <b:Quantity definition="urn:ogc:def:property:OGC:pressure">
    <c:name>pressure</c:name>
    <b:uom>
      {
      doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Outputs/Preassure/uom/text()
      }
    </b:uom>
    <b:constraint>
```

```
                        <b:AllowedValues>
                           <b:interval>
                              {
                              doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Outputs/Preassure/interval/text()
                              }
                           </b:interval>
                        </b:AllowedValues>
                     </b:constraint>
                     <b:quality>
                        <b:Quantity>
                           <b:value>
                              {
                              doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Outputs/Preassure/value/text()
                              }
                           </b:value>
                        </b:Quantity>
                     </b:quality>
                  </b:Quantity>
               </a:output>
            </a:OutputList>
         </a:outputs>
         <a:components>
            <a:ComponentList>
               <a:component>
                  <a:Component
id="{doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Components/ComponentList/Component/Id}">
                     <c:description>
                        {
                        doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Components/ComponentList/Component/Description/text()
                        }
                     </c:description>
                     <a:inputs>
```

```
                        <a:InputList>
                           <a:input
name="{doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Components/ComponentList/Component/ComponentInputs/ComponentIn
put/Name}">
                              {
                              doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Components/ComponentList/Component/Inputs/InputList
/Input/text()
                              }
                              <a:ObservableProperty
definition="{doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Components/ComponentList/Component/ComponentInputs/Compon
entInput/Property}"/>
                           </a:input>
                        </a:InputList>
                     </a:inputs>
                     <a:outputs>
                        <a:OutputList>
                           <a:output
name="{doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Components/ComponentList/Component/OutputsComponent/OutputCom
ponent/Name}">
                              <c:Quantity>
                                 <c:uom>
                                    {
                                    doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Components/ComponentList/Component/Outputs/O
utputList/Output/Quantity/uom/text()
                                    }
                                    {
                                    doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Components/ComponentList/Component/OutputsCo
mponent/OutputComponent/UOM/text()
                                    }
                                 </c:uom>
                                 <c:value>
                                    {
```

```
                                    doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Components/ComponentList/Component/Outputs/O
utputList/Output/Quantity/value/text()
                                    }
                                    {
                                    doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Components/ComponentList/Component/OutputsCo
mponent/OutputComponent/Value/text()
                                    }
                                 </c:value>
                              </c:Quantity>
                           </a:output>
                        </a:OutputList>
                     </a:outputs>
                     <b:parameters>
                        <b:ParameterList>
                           <b:parameter
name="{doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Components/ComponentList/Component/Parameters/Parameter/Name}">
                              <a:QuantityRange
definition="{doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Components/ComponentList/Component/Parameters/Parameter/Data/
Field/Definition}">
                                 <c:name>
                                    {
                                    doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Components/ComponentList/Component/Parameters
/Parameter/Data/Field/UOM/text()
                                    }
                                    {
                                    doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Components/ComponentList/Component/Parameters
/Parameter/Data/Field/Name/text()
                                    }
                                 </c:name>
                                 <a:uom>
                                    {
                                    doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Components/ComponentList/Component/Parameters
```

```
/Parameter/Data/Field/UOM/text()
                                    }
                                </a:uom>
                                <a:value>
                                    {
                                    doc('file:///c:/Teds2SML/ReadTEDSHTTPResponse.xml')/ReadTEDSHTTPResponse/teds/Components/ComponentList/Component/Parameters
/Parameter/Data/Field/Value/text()
                                    }
                                </a:value>
                            </a:QuantityRange>
                        </b:parameter>
                    </b:ParameterList>
                </b:parameters>
            </a:Component>
        </a:component>
    </a:ComponentList>
</a:components>
</a:System>
</a:member>
</a:SensorML>
```

9.5 Recommendation

For the units harmonization the implementation can provide a reference to a unit of measure that is accessible externally or within the current XML file which defines the unit of measure according to schema. As this option is indirectly stated in the SML specification, and does not appear in the 1451.0 standard for general units of measurements (it does for CRS), we invite both specifications to advise the above for machine-readable unit description across the two standards, independently of naming conventions, which should preferably be using UCUM.

For the general harmonization issue this group is still working on finalizing the details and results will be published in the following months.

10. Acknowledgements

The Oceans IE Team greatly appreciates the contributions of all the participants at OOSTethys. Some of them are listed as contributors if this document (section 1.2). Other contributors to this activity include: Ben Domenico (UNIDATA/UCAR), Bill Howe (NANOOS), David Coyle (USGS), Mark Wholey (ASA), Matthew Howard (TAMU), Jeremy Cothran (USC), Richard P. Signell (USGS), Paul Daisey (Image Matters LLC), and Jeff deLaBeaujardiere (NOOA/IOOS).

This activity has been supported by the National Science Foundation under award number ATM-0447031; Southeastern Universities Research Association (SURA) Monterey Bay Aquarium Research Institute and the David and Lucile Packard Foundation; Office of Naval Research, Award N00014-04-1-0721; and, by NOAA Ocean Service Award NA04NOS4730254. Any Opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

REFERENCES

Bermudez L.E., E. Delory, T.C. O'Reilly, and J.D.R. Fernandez, “Ocean Observing Systems Demystified,”, Oceans'09 MTS/IEEE Biloxi, MS, 2009.

O’Reilly T., Headley, et al. “Instrument interfaces for interoperable sensor networks”, IEEE OCEANS 2009, May 2009.

O’Reilly T., K. Headley et al, “MBARI technology for self-configuring interoperable ocean observatories”, IEEE OCEANS 2006, 18-21 Sept. 2006, Page(s):1 – 6, Digital Object Identifier 10.1109/OCEANS.2006.306893.

Song E., K. Lee, “Smart transducer Web services based on IEEE 1451.0 standard”, IMTC 2007-Instrumentation and Measurement Technology Conference. WARSAW, POLAND, MAY 1-3, 2007, pp.1-6.